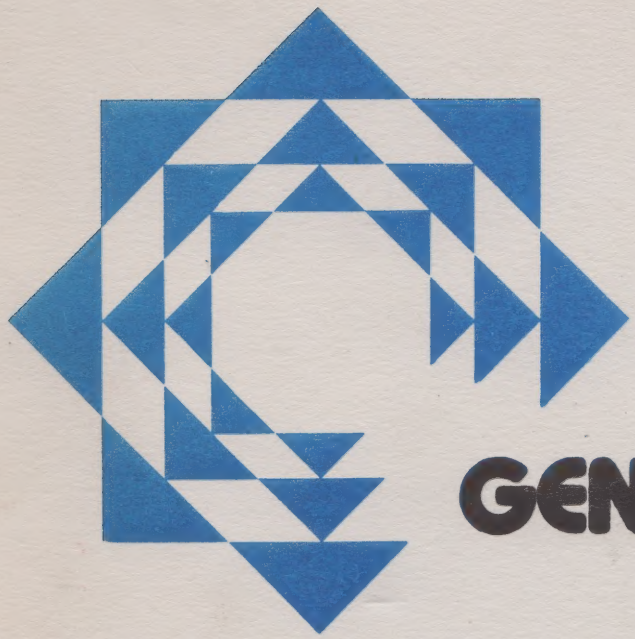

GENERAL ROBOTICS
CORPORATION

ROBOT CONTROL LANGUAGE™ WITH SAVVY®

ROBOT CONTROL LANGUAGE(tm) WITH SAVVY(R)

PROGRAMMER'S MANUAL

GENERAL ROBOTICS CORPORATION Golden, Colorado

1986

GENERAL ROBOTICS CORPORATION
14618 W. 6th Avenue, Suite 150
Golden, CO 80401
(303) 277-1574 or 277-1575
(800) 422-4265

ROBOT CONTROL LANGUAGE WITH SAVVY
PROGRAMMER'S MANUAL

TABLE OF CONTENTS

Figures

A. OVERVIEW

Robot Control Language(tm) with Savvy(R) -- RCL(tm) for short -- provides a simple, straightforward method of programming the RB5X(tm) to perform many functions without using the terse Tiny BASIC code that ultimately controls the robot's actions.

RCL users instruct the RB5X using "robot tasks" like MOVE FORWARD, GO CLOCKWISE, TURN THE WRIST, TURN ON THE HORN, etc. RCL then automatically cross-compiles the robot task, line by line, into Tiny BASIC code and then downloads the code into the robot's on-board memory by means of an RS-232 serial communications interface.

In addition to commands that initiate and control RB5X's action, RCL also includes facilities for error trapping and compilation of loops. Each element of RCL is as meaningful to Savvy as any Savvy primary language command such as ADD, EDIT, or SAVE. Furthermore, the system is flexible, to allow the addition of new RCL commands, and the editing and modification of all RCL elements and associated Savvy tasks. Since RCL is Savvy-based, all robot vocabulary can be extended with synonyms for existing commands.

This document presumes the reader has gone through the RCL Tutorial, has reviewed the Savvy Personal Language Reference Folio, has fundamental Savvy task development skills, and possesses at least a superficial knowledge of NSC Tiny BASIC. (If you need more information about how to use Tiny BASIC, consult the RB5X Reference Manual.)

B. STRUCTURE

Accompanying this document is a complete directory of all RCL vocabulary. The vocabulary has four main elements: action commands, programming commands, parameter-passing functions, and error-checking and compiler routines.

ACTION COMMANDS

These commands instruct the robot to perform an activity.

Examples:

```
FOLLOW TAPE
GO CLOCKWISE
MOVE BACKWARD
```

PROGRAMMING COMMANDS

These commands are used for initialization, error checking, looping, and other programming conventions.

Examples:

```
LOAD
REPORT ASSIGN ERROR
REPEAT THIS LOOP
```

PARAMETER-PASSING FUNCTIONS

These functions pass specific parameters to the robot, sometimes in the form of distances to move, degrees to rotate, phrases to speak, etc. Other parameter-passing functions are used strictly as programming conventions, such as counts for a loop.

Examples:

```
Parameter-passing functions as robot
instructions:

MOVE DISTANCE BACKWARD for this many feet: <1>
MOVE TIMED FORWARD for this many seconds: <1>

Parameter-passing functions for programming:

DEFINE A VARIABLE called <1>
REPEAT THIS COUNTED LOOP for the counter <1>
```

ERROR-CHECKING AND COMPILER ROUTINES

All RCL vocabulary elements that are preceded by a 'Z' are for error checking and compilation.

Examples:

```
Z ERROR CHECK MATH FUNCTION
Z LOOK FOR MATCHING STATEMENT
Z LOAD SONAR
```

There are several elements in the directory preceded by an 'X'. These are sample RCL robot tasks (i.e., X Simple Simon).

Each element of the RCL vocabulary -- action commands, programming commands, parameter-passing functions, and the error-checking and compiler routines -- has a corresponding Savvy task or function that actually executes the task. These tasks and functions may also call other tasks and functions, and may in turn be called by other tasks. Their operation is completely transparent to the end-user. (Appendix 2 contains the complete text of all RCL tasks and functions.)

VOCABULARY ELEMENTS IN ROBOT TASKS

A user composing a robot task selects action commands, programming commands, and parameter-passing functions to form a robot task, which is also an actual Savvy task. The task can be edited with the Savvy editor and is saved on disk just like any other Savvy task. The task name becomes part of RCL. The X FIND CHARGER ROUTINE listed as Sample 1 is an example of a simple task. It has looping, parameter passing, and other programming conventions.

Sample 1

```
X CHARGER FINDER (a Task)

1 Does WAIT this many seconds 20
2  and PREPARE THE ROBOT
3  and INITIALIZE MEMORY
4  and BEGIN A LOOP
5  and PREPARE THE ROBOT
6  and CLEAR ALL ITEMS
7  and BEGIN A LOOP
8  and MOVE FORWARD
9  and EXIT IF ANY BUMPER TOUCHED
10  and EXIT IF THE TAPE IS SENSED
11  and REPEAT THIS LOOP
12  and TEST IF the variable "R" is (=,<,>,<>) "=" to 0
13  and EXIT THIS LOOP
14  and END HERE
15  and MOVE WITH BETA INTELLIGENCE
16  and REPEAT THIS LOOP
17  and FOLLOW TAPE
18  and MAINTAIN CHARGE
19  and END  (Task is 7% full.)
```


From this set of instructions, RCL compiles the Tiny BASIC text necessary to accomplish the specified activity or function. The error-checking and compiler routines are used as necessary in the actual Savvy tasks that correspond to RCL elements.

RCL AND ITS RELATIONSHIP TO TINY BASIC

Programming the RB5X in RCL first requires communication between two different hardware configurations, the 6502-based Apple II and the INS8073-based robot. The diagram in Figure 1 illustrates the hierarchy of languages that perform this communication and their relationship. (FORTH is the machine language that interfaces Savvy with the 6502.)

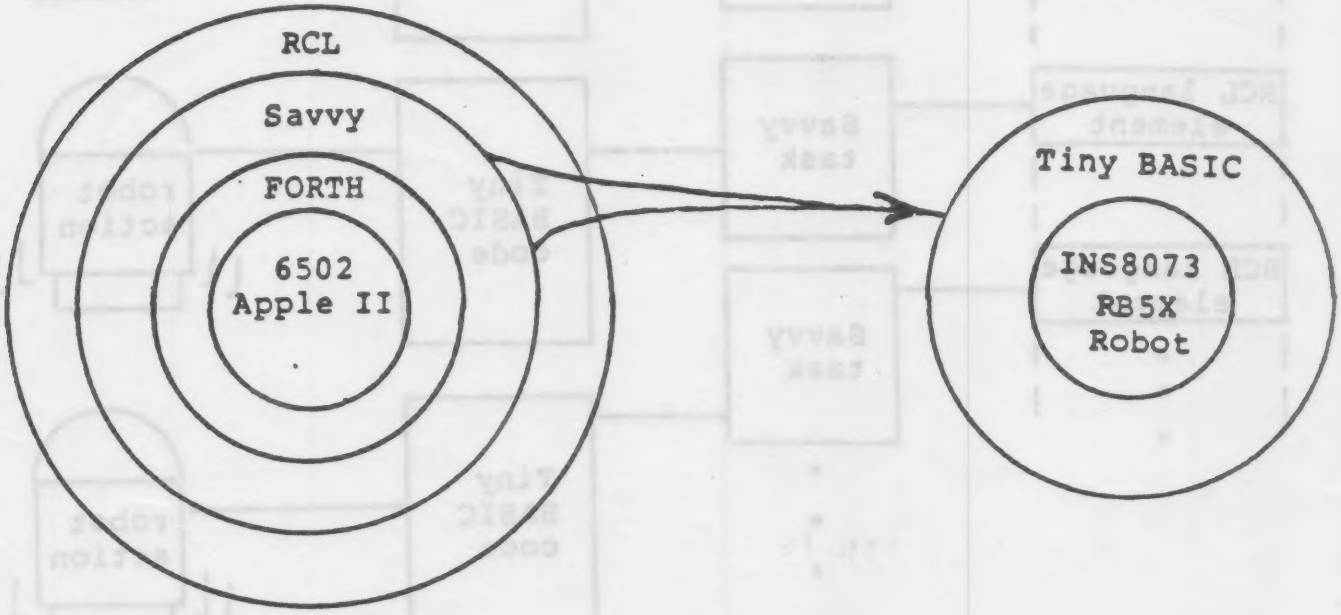

Figure 1.
Hardware and Language Relationships

As illustrated in Figure 1, Tiny BASIC communicates with the robot hardware. Because of its machine code-like nature, many Tiny BASIC instructions are necessary to implement an RCL command. Figure 2 illustrates the language structures and relationships.

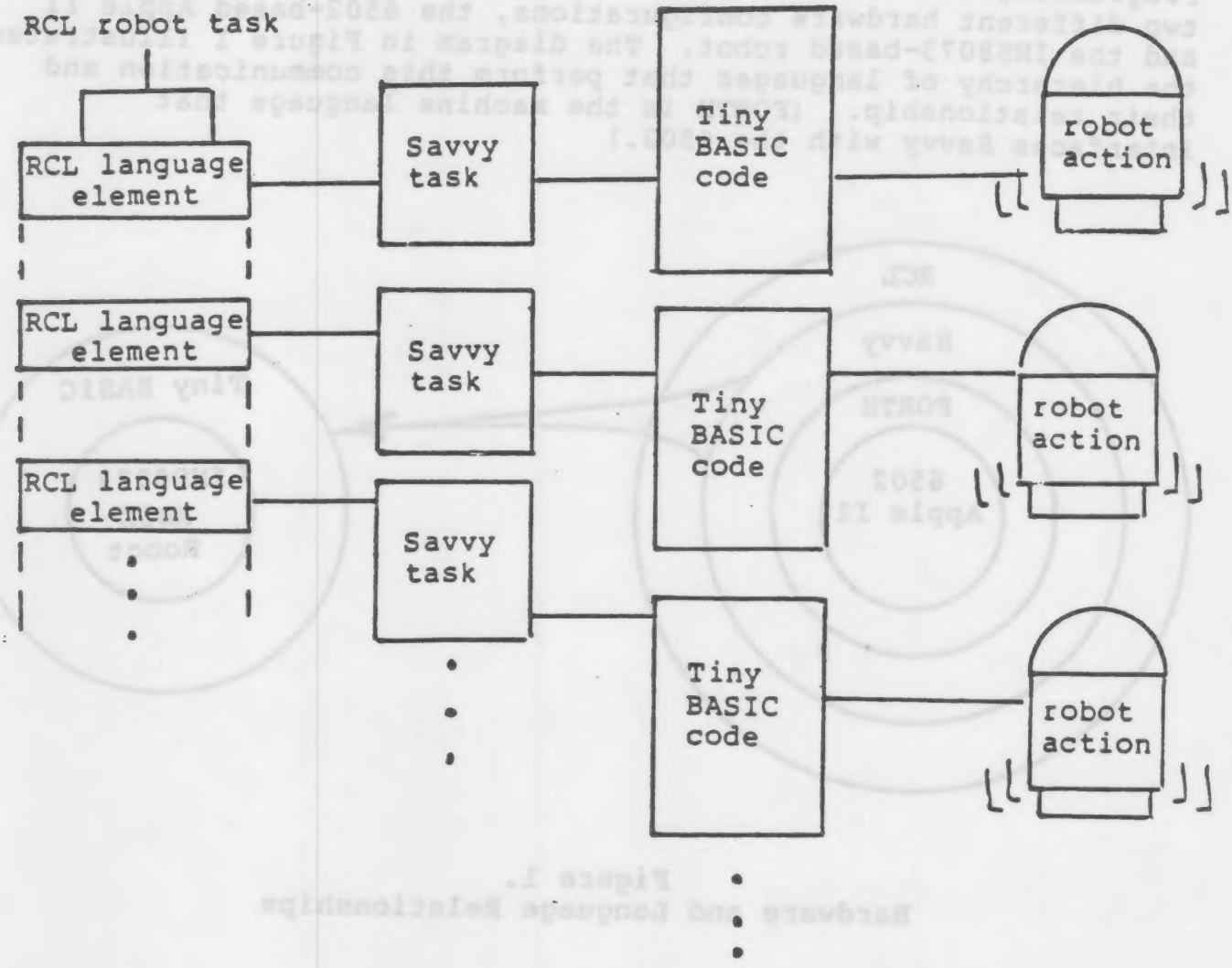

Figure 2.
Language Structures and Relationships

COMPILING RCL INTO TINY BASIC

The Tiny BASIC program text is stored in a Savvy folder named SOURCE. It is indexed (keyed) by the item STATEMENT NUMBER and the two items BASIC TEXT and MEANING. BASIC TEXT is the Tiny BASIC text, a combination of symbols and numbers, such as @#7803=#98, that specify action, define addresses, turn bits on and off, etc. MEANING is the documentary description that accompanies the code.

The folder structure for stored Tiny BASIC text in RCL is:

```
SOURCE (a folder)

  1 Id STATEMENT NUMBER
  2 Item BASIC TEXT
  3 Item MEANING
```

A Savvy task and function perform the compilation according to the following process before the data is actually stored in the SOURCE folder.

1. Increment the current line number to form the STATEMENT NUMBER for this statement
2. Copy the appropriate piece of text to the BASIC TEXT
3. Copy any textual description to MEANING
4. Save this page of data in the SOURCE folder

The following tasks compile RCL according to the preceding algorithm:

Sample 2

```
Z COMPILE the BASIC statement <1> which means <2> (a Function)

 1 Does Z INCREMENT STATEMENT NUMBER
 2  and COPY from <1> to BASIC TEXT
 3  and COPY from <2> to MEANING
 4  and SAVE new page in SOURCE
 5  and END  (Task is 2% full.)


Z INCREMENT STATEMENT NUMBER (a Task)

 1 Does ADD the STATEMENT NUMBER and STATEMENT NUMBER INCREMENT
   VALUE
 2   and COPY from SUM to STATEMENT NUMBER
 3   and END  (Task is 1% full.)
```


C. IMPLEMENTATION

RCL is flexible and extensive enough to allow users to program the RB5X to perform countless activities. The best way to understand how the RB5X works and how robot tasks are composed is to actually key one in. By entering the following robot task (Sample 3), you learn how to hook up the robot, how to prepare its memory, how to download the robot task from your Apple to your RB5X, and other aspects of "training" the RB5X.

INSTRUCTIONS FOR KEYING IN THE SAMPLE TASK

Composing a robot task is just like defining a regular Savvy task. (Again, these instructions assume you have a basic familiarity with Savvy and understand the way Savvy prompts for input.) In the following dialogue, the user's response is in upper case and Savvy's prompts are in lower case, just as in actual use if you have an 80-column card or an Apple IIe.

1. Begin by defining a task called TEST PROGRAM.

   DEFINE a task called TEST PROGRAM

2. Type the text, beginning with line 1, just as it is listed in Sample 3 on page C-2. Remember that all lower case words in the script are Savvy prompts that RCL automatically inserts after you press the RETURN key.

3. Check the task for errors and correct with the Savvy editor.

4. If the robot task appears satisfactory, instruct Savvy to BUILD the source task. Savvy prompts for the task name.

   BUILD a program from the robot task "TEST PROGRAM"

   First the system displays the following message:

   Would you like your robot task checked for errors? (Y/N).

   After your response, the system displays the message:

   Now building robot task TEST PROGRAM

   ...one moment, please.

   To see the Tiny BASIC code that the BUILD command creates, type LIST THE PROGRAM and you see the Tiny BASIC with comments.

5. Plug the Apple into the RB5X via the RS-232 serial interface and switch the robot on. Type:

   LOAD

   The drive spins and RCL displays the message:

   ** Now loading...One moment please **

   The system then displays the following message:

   Now loading your robot task...###

6. After RCL downloads the completed robot task, the RB5X executes the TEST PROGRAM. If the drives do not spin and the robot does not respond, the baud rate may be incorrect (it must be set at the same rate on both robot and serial interface), the serial card may not be connected properly, or the robot may be inadequately charged.

7. To run again, type:

   RUN<RET>

Sample 3

```
TEST PROGRAM (a Task)

 1 Does PREPARE THE ROBOT
 2  and TURN ON THE FLASHING LIGHTS
 3  and MOVE DISTANCE FORWARD for this many feet 2
 4  and SPIN LEFT 90 DEGREES
 5  and SPIN RIGHT 90 DEGREES
 6  and MOVE TIMED BACKWARD for this many seconds 5
 7  and PIVOT ON LEFT CLOCKWISE
 8  and WAIT this many seconds 4
 9  and PIVOT ON LEFT COUNTERCLOCKWISE
10  and WAIT this many seconds 4
11  and PIVOT ON RIGHT CLOCKWISE
12  and WAIT this many seconds 4
13  and PIVOT ON RIGHT COUNTERCLOCKWISE
14  and WAIT this many seconds 4
15  and STOP ALL MOTION
16  and HONK the horn for this many seconds 2
17  and END  (Task is 8% full.)
```

Remember that you type Q<RET> to complete the robot task and then RCL replies with the END line.

WRITING NEW ROBOT TASKS

As mentioned in the preceding instructions, creating a new robot task is just like defining a new Savvy task. However, instead of using Savvy commands and primaries directly, RCL is used to specify instructions to the robot. Each robot task must begin with the directive:

```
PREPARE THE ROBOT
```

This initializes the RB5X hardware and prepares it to accept the robot task that has been compiled into Tiny BASIC. After the PREPARE THE ROBOT command has been entered, other RCL language elements can then be combined to define a robot task.

Simple robot tasks such as the TEST PROGRAM illustrated in Sample 3 do not include loops or conditional tests, nor do they require the use of the special compiler routines. Other, more complicated robot tasks use these conventions. When composing robot tasks with looping conventions, remember that loops may be nested no more than eight deep in Tiny BASIC.

Sample 4 is a robot task that is more complex than TEST PROGRAM and illustrates looping conventions. Try it out to begin to get a feel for the way the language works.

Sample 4

```
TEST LOOPS (a Task)

 1 Does CALL this line "START OVER"
 2  and PREPARE THE ROBOT
 3  and BEGIN A LOOP
 4  and DEFINE A VARIABLE called "L"
 5  and EXIT IF ANY BUMPER TOUCHED
 6  and REPEAT THIS LOOP
 7  and BEGIN A COUNTED LOOP called "L" beginning
       at 1 ending at 5
 8  and TURN ON THE FLASHING LIGHTS
 9  and WAIT this many seconds 1
10  and TURN OFF THE FLASHING LIGHTS
11  and wait this many seconds 1
12  and REPEAT THIS COUNTED LOOP for the counter "L"
13  and BEGIN A LOOP
14  and MOVE FORWARD
15  and EXIT IF SONAR distance value is less than 95
16  and REPEAT THIS LOOP
17  and MOVE BACKWARD
18  and SET the variable "L" equal to 0
19  and BEGIN A LOOP
20  and CALCULATE variable "L" = "L" (+,-,*,/) "+" 1
21  and HONK the horn for this many seconds 1
22  and WAIT this many seconds 1
23  and TEST IF the variable "L" is (=,<,>,<>) "=" compared
       to 4
24  and REPEAT THIS LOOP
25  and STOP ALL MOTION
26  and JUMP to the line called "START OVER"
27  and END HERE
28  and END  (Task is 19% full.)
```

After new tasks are written, you may go through the BUILD procedure described in the section on "Instructions for Keying in the Sample Task." However, in certain cases you may want to skip the two-part BUILD and LOAD procedure and just type:

```
BUILD AND LOAD
```

This RCL primary task performs the BUILD and the LOAD operations consecutively after the robot task title is entered, without querying again. The BUILD AND LOAD routine does not, however, have the same error-checking faculties as the separate BUILD and LOAD tasks, and we do not advise using it on new tasks.

EXTENDING RCL AND DEFINING NEW ACTIONS

Extending RCL and defining new actions, either to expand the language or to accommodate new hardware, requires a knowledge of Tiny BASIC. The proper Tiny BASIC code must be inserted in the source folder to provide communication with the hardware.

Implementing new RCL primary tasks begins with design at the Tiny BASIC level. When the fundamental instructions are coded, they may be incorporated into Savvy tasks. Certain existing RCL elements make it easy to write one-liners. These are Z INCREMENT STATEMENT NUMBER and COMPILE the BASIC statement <1> which means <2>. (The text of these routines is listed in the section "Compiling RCL into Tiny BASIC.")

For example, a task such as STOP ALL MOTION is done like this:

STOP ALL MOTION (a task)

1 Does COMPILE the BASIC statement "@#7802=0" which means "STOP ALL MOTION"

Some new primary tasks require parameterized subroutines. For example, if you were implementing a siren, you would write a routine similar to TURN ON THE HORN listed in Sample 5.

The first line compiles the Tiny BASIC code; the second is a subroutine to turn on a bit. It is necessary to ensure that the target subroutine is there, so a task is Z INSURE PORT 7801 SUBROUTINE IS. The text of this task is listed as Sample 6.

Sample 5

TURN ON THE HORN (a Task)

1 Does Z COMPILE the BASIC statement "X=#80" which means "TURN ON HORN"
2 and Z COMPILE the BASIC statement "GOSUB 3100" which means "GO TURN ON A BIT"
3 and Z INSURE PORT 7801 SUBROUTINE IS
4 and END (Task is 6% full.)

Sample 6

```
Z INSURE PORT 7801 SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE DOES NOT CONTAIN 3,100 then
 2  Do COPY from STATEMENT NUMBER TO HERE
 3  and Z INCLUDE at 3,100 the statement "U=@#7801"
        which means "TURN ON A BIT"
 4  and Z INCLUDE at 3,105 the statement "U=U OR X"
        which means ""
 5  and Z INCLUDE at 3,110 the statement "@#7801=U"
        which means ""
 6  and Z INCLUDE at 3,115 the statement "RETURN"
        which means ""
 7  and Z INCLUDE at 3,120 the statement "U=@#7801"
        which means "TURN OFF A BIT"
 8  and Z INCLUDE at 3,125 the statement "U=U AND X"
        which means ""
 9  and Z INCLUDE at 3,130 the statement "@#7801=U"
        which means ""
10  and Z INCLUDE at 3,135 the statement "RETURN"
        which means ""
11  and COPY from HERE to STATEMENT NUMBER
12  and END of test
13  and END  (Task is 23% full.)
```

Since RCL is Savvy-based, the existing commands can be "extended with synonyms" using the Savvy ASSOCIATE command. These might be abbreviations of certain frequently used commands to speed up data entry, such as TOH for TURN ON THE HORN. And, because of Savvy's Adaptive Pattern Recognition Processing (APRP), the more synonyms added to RCL, the more Savvy is able to "help out" in cases of misspelling or improperly worded phrases.

MODIFYING EXISTING RCL TASKS

The Savvy tasks that actually implement the RCL commands may be modified to suit your needs.

For example: If you find it tiresome to begin each robot task with PREPARE THE ROBOT, you can insert the Tiny BASIC initialization code (@#7803=#98) in the LOAD routine so that RCL automatically does this initialization. If you don't like using periods as delimiters in the phonemes, you can change them to slashes (/) by instructing Savvy to COPY from "/" to DELIMITER. You can change display messages with Savvy's editor or modify the RUN task to allow you to communicate "interactively" with the robot. (Add wait states so you must type RUN at the keyboard after each command as a useful debugging facility.)

Of course, before you begin modifiying the Savvy tasks, be sure you have a recent backup of your disk so you can restore the task to its original state if there are any problems.

D. DEBUGGING AND ERROR TRAPPING

Debugging and error trapping in new robot tasks and primary tasks takes place at three levels:

Robot task level (RCL level)
Savvy task level
Tiny BASIC level

At the highest level, the robot task level, there are error-trapping routines in Savvy itself that check new scripts for language errors, matching statement errors, errors of correspondence between arguments and functions, etc. These routines do not check for logic errors. When RCL detects an error, it saves the error and displays the number of the offending line.

Some errors that are returned may not interfere with the purpose of the script. In fact, some errors returned may be things you intended to do. Endless loops are usually considered a bug in most programming situations; however, endless loops to carry on a specified motion may be something you intend the RB5X to do.

When a Savvy script is modified and then does not function as intended, check all other routines that are called by and from the script to check for errors. Because of the reentrant nature of these tasks, bugs may not necessarily show up in the modified task. If an unmodified task that previously ran without problems begins to behave in a peculiar fashion, check to see if the tasks it calls or is called by have been recently modified.

Debugging new tasks that extend RCL must begin at the Tiny BASIC level. Be sure the Tiny BASIC code works before introducing it into the task that compiles it. If it works at that level, test at the Savvy task level, and finally at the robot task level. Debugging Tiny BASIC is not an activity for the casual or inexperienced programmer.

E. MISCELLANEOUS PROGRAMMER'S NOTES

Listed below are various notes concerning language use and conventions and other programming information.

1. JUMP to the line called <1>

   CALL this line <1>

   These may be nested in any loop and the label name must be in quotes.

2. MOVE THE ARM FROM THE SHOULDER (UP, DOWN, IN, OUT): <1> for this many degrees: <2>

   MOVE THE FOREARM (IN,OUT): <1> for this many degrees: <2>

   MOVE THE HAND (OPEN, CLOSE): <1> for this (1-100) %:<2>

   TURN THE WRIST (LEFT, RIGHT): <1> for this many degrees:<1>

   The first argument in all of these functions does not have to be enclosed in quotation marks. If the second argument is incorrect, a warning message appears on the screen and the BUILD stops.

3. PREPARE THE VOICE -- Initializes the voice

4. LOAD the phrase called <1> with phonemes <2>

   For example:

   ```
   LOAD THE PHRASE called "HELLO" with the
   phonemes "H.E1....STOP."
   ```

   Each phoneme must have a period after it, including the last phoneme in the sequence. There can be no spaces in between. If any periods are omitted, the machine goes into an endless loop. The last phoneme must always be STOP.

   The "LOAD THE PHRASE" action should be near the top of your robot task, but after the "PREPARE THE VOICE" command. You should not include the "LOAD THE PHRASE" task with any of your loops since it only needs to be done once. Use the "SPEAK the phrase called <1>" in the body of your task when you want the robot to speak the phrase you previously loaded.

5. In the following tasks:

   DEFINE A VARIABLE

   BEGIN A COUNTED LOOP

   REPEAT THIS COUNTED LOOP

   CALCULATE

   REPEAT A LIMITED LOOP

   TEST IF

   the variable used must be a legal Tiny BASIC variable (A-Z).

6. CALCULATE

   "CALCULATE" allows the programmer to alter a Tiny BASIC variable by operating upon it with one of the four standard arithmetic functions (add, subtract, multipy, divide) along with either another variable or a constant value. A common use of this task is to increment a variable for use in a loop. For example:

   CALCULATE variable "A" = "A" "+" 1

   In this case "A" is the variable to be operated on. The variable is in quotation marks because Savvy must treat the Tiny BASIC variable as a literal. The numeral, on the other hand, does not require quotes since it is a constant in both Savvy and in Tiny BASIC. Likewise, the operand must be placed in quotes since Savvy does not recognize it as a constant.

7. TEST IF

   "TEST IF" gives the programmer the option of being able to test a variable against another variable or a constant value while in a loop. The result of such a test will be either true or false. If the result is true, then the loop continues; if the result is false, the task branches out of the loop statement immediately following the end of the loop. For example:

```
BEGIN A LOOP
   CALCULATE variable "A" = "A" (+,-,*,/) "+" 1
   TEST IF the variable "A" is (=,<,>,<>) "=" compared
   to 4
REPEAT THIS LOOP
END HERE
```


In this procedure, the loop will continue until the variable A is equal to four; when this occurs, the task terminates.

8. When using variables in your tasks, such as for the TEST IF task, the CALCULATE task, or the COUNTED LOOPS, be careful about using the following variables; they are already being used within some of the RCL primary tasks:

   A, D, I, J, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

   There may or may not be interaction between your choice of variable and the use of that variable within one of the tasks, so it is best to use other variable names unless you check the tasks being used, or look at the LIST THE PROGRAM.

F. APPENDICES

APPENDIX 1

Provided RCL Tasks

```
PICK A RANDOM DIRECTION (a Task)

 1 Does Z COMPILE the BASIC statement "GOSUB 3040" which means "PICK RANDOM DI
CTION"
 2  and Z INSURE RANDOM TURN SUBROUTINE
 3  and END   (Task is 4% full.)

PIVOT ON LEFT CLOCKWISE (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#04" which means "RIGHT REVERSE"
 2  and END   (Task is 3% full.)

PIVOT ON LEFT COUNTERCLOCKWISE (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#08" which means "RIGHT FORWARD "
 2  and END   (Task is 3% full.)

PIVOT ON RIGHT CLOCKWISE (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#01" which means "LEFT FORWARD"
 2  and END   (Task is 3% full.)

PIVOT ON RIGHT COUNTERCLOCKWISE (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#02" which means "LEFT REVERSE"
 2  and END   (Task is 3% full.)

PREPARE THE ROBOT (a Task)

 1 Does Z COMPILE the BASIC statement "@#7803=#98" which means "INITIALIZE I/O
 2  and END   (Task is 3% full.)

PREPARE THE VOICE (a Task)

 1 Does Z COMPILE the BASIC statement "@#780B=#A7:S=TOP" which means "PREPARE T
E VOICE"
 2  and COPY from "YES" to VOICE NEEDED
 3  and END   (Task is 5% full.)

REMARK that: <1>  (a Function)

 1 Does PASTE the "REM " in front of <1>
 2  and Z COMPILE the BASIC statement FRONT which means "REMARK"
 3  and END   (Task is 2% full.)
```

```
REPEAT THE LIMITED LOOP unless <1> is (<,>,=,<>) <2> to <3>  (a Function)

 1 Does Z ERROR CHECK IF/UNTIL STATEMENT
 2  and PASTE the "UNTIL " in front of <1>
 3  and PASTE the FRONT in front of <2>
 4  and PASTE the FRONT in front of <3>
 5  and Z COMPILE the BASIC statement FRONT which means "CHECK EXIT CONDITION"
 6  and COPY from "UNTIL" to STRUCTURE TYPE
 7  and SAVE new page in STRUCTURES
 8  and END   (Task is 7% full.)

REPEAT THIS COUNTED LOOP for the counter <1>  (a Function)

 1 Does Z DO CHECK if item <1> is in folder "VARIABLES"
 2  and IF the RESULT  IS  "FAIL" then
 3   Do    Z SAVE FOLDER ERROR
 4  and OTHERWISE
 5   Do    GET the page indexed by <1> in folder VARIABLES
 6  and    COPY from VAR NAME to <1>
 7  and    END of test
 8  and PASTE the "NEXT " in front of <1>
 9  and Z COMPILE the BASIC statement FRONT which means "REPEAT THE COUNTED LOO
(FOR/NEXT)"
10  and COPY from "NEXT" to STRUCTURE TYPE
11  and SAVE new page in STRUCTURES
12  and END   (Task is 11% full.)

REPEAT THIS LOOP (a Task)

 1 Does COPY from STATEMENT NUMBER to HERE
 2  and Z GET PREVIOUS BEGIN
 3  and DELETE the page indexed by STATEMENT NUMBER from folder LOOPS
 4  and PASTE the "GOTO " in front of STATEMENT NUMBER
 5  and COPY from HERE to STATEMENT NUMBER
 6  and Z COMPILE the BASIC statement FRONT which means "REPEAT THIS LOOP"
 7  and Z COMPILE the BASIC statement "REM EXIT TO HERE" which means ""
 8  and Z PATCH UP EXIT STATEMENTS
 9  and COPY from "REPEAT" to STRUCTURE TYPE
10  and SAVE new page in STRUCTURES
11  and END   (Task is 10% full.)

REPORT ASSIGN ERROR (a Task)

 1 Does CARRIAGE return
 2  and IF the ASSIGNMENT  IS  "BUILD" then
 3   Do    DISPLAY the "I'm sorry, I could not find the requested robot script.

 4  and    HALT and return to Ready
 5  and OTHERWISE
 6   Do    DISPLAY the "REPORT ASSIGN ERROR!"
 7  and    END of test
 8  and END   (Task is 10% full.)
```

```
REPORT DELETE ERROR (a Task)

 1 Does IF the ASSIGNMENT  IS  "BUILD" then
 2   Do    DISPLAY the ""
 3  and OTHERWISE
 4   Do    DISPLAY the "REPORT DELETE ERROR!"
 5  and    END of test
 6  and END   (Task is 4% full.)

REPORT GET ERROR  (Undefined) (a Task)

Undefined task!   (Task is 1% full.)

REPORT LOAD ERROR (a Task)

 1 Does CARRIAGE return
 2  and DISPLAY the "Sorry, there is not a program to load."
 3  and HALT and return to Ready
 4  and END   (Task is 4% full.)

REPORT MATH ERROR (a Task)

 1 Does DISPLAY the ""
 2  and END   (Task is 1% full.)

REPORT REPLACE ERROR (a Task)

 1 Does END   (Task is 1% full.)

REPORT SAVE ERROR (a Task)

 1 Does CARRIAGE return
 2  and DISPLAY the "REPORT SAVE ERROR!"
 3  and END   (Task is 2% full.)

RUN (a Task)

 1 Does PRINT the ""
 2  and OUTPUT from "RUN" using format DEFAULT
 3  and CARRIAGE return
 4  and END   (Task is 1% full.)

SET the variable <1> equal to <2>  (a Function)

 1 Does Z DO CHECK if item <1> is in folder "VARIABLES"
 2  and IF the RESULT  IS  "FAIL" then
 3   Do    Z SAVE FOLDER ERROR
 4  and OTHERWISE
 5   Do    GET the page indexed by <1> in folder VARIABLES
 6  and    COPY from VAR NAME to <1>
 7  and    END of test
 8  and Z DO NUMBER TEST on <2>
 9  and IF the RESULT  IS  "FAIL" then
10   Do    Z DO CHECK if item <2> is in folder "VARIABLES"
11  and    IF the RESULT  IS  "FAIL" then
12   Do       Z SAVE NUMBER OR FOLDER ERROR
13  and    OTHERWISE
14   Do       GET the page indexed by <2> in folder VARIABLES
15  and       COPY from VAR NAME to <2>
16  and       END of test
```

```
SET the variable <1> equal to <2>  (a Function)

17  and    END of test
18  and PASTE the <1> in front of "="
19  and PASTE the FRONT in front of <2>
20  and Z COMPILE the BASIC statement FRONT which means "INITIALIZE"
21  and END   (Task is 15% full.)

SHOW ERRORS (a Task)

 1 Does CLEAR the page
 2  and OUTPUT from "LINE NUMBER" using format DEFAULT
 3  and OUTPUT from SPACES using format 23
 4  and OUTPUT from "ERROR MESSAGE" using format DEFAULT
 5  and CARRIAGE return
 6  and CARRIAGE return
 7  and BEGIN at page indexed by FIRST in folder ERRORS
 8  and START a loop
 9   Do    GET the page indexed by NEXT in folder ERRORS
10  and    IF the PAGE  IS  EMPTY then
11   Do       LEAVE this loop
12  and       END of test
13  and    IF the ERROR NUMBER  IS  200 then
14   Do       CARRIAGE return
15  and       OUTPUT from SPACES using format 25
16  and       OUTPUT from "--- Structure Errors ---" using format DEFAULT
17  and       CARRIAGE return
18  and       CARRIAGE return
19  and       END of test
20  and    OUTPUT from LINE NUMBER using format 7
21  and    OUTPUT from SPACES using format 8
22  and    OUTPUT from ERROR MESSAGE using format DEFAULT
23  and    CARRIAGE return
24  and    REPEAT this loop
25  and CARRIAGE return
26  and END   (Task is 15% full.)

SHOW THE PHONEME DICTIONARY (a Task)

 1 Does CLEAR the page
 2  and OUTPUT from "Phoneme Symbol" using format DEFAULT
 3  and OUTPUT from SPACES using format 10
 4  and OUTPUT from "Phoneme Code" using format DEFAULT
 5  and CARRIAGE return
 6  and CARRIAGE return
 7  and BEGIN at page indexed by FIRST in folder PHONEMES
 8  and START a loop
 9   Do    GET the page indexed by NEXT in folder PHONEMES
10  and    IF the PAGE  IS  EMPTY then
11   Do       LEAVE this loop
12  and       END of test
13  and    OUTPUT from SPACES using format 5
14  and    OUTPUT from PHONEME ID using format 24
15  and    OUTPUT from PHONEME CODE using format 2
16  and    CARRIAGE return
17  and    REPEAT this loop
18  and END   (Task is 10% full.)
```

```
SPEAK the phrase called:  <1>  (a Function)

 1 Does BEGIN at page indexed by FIRST in folder SPEECH LOAD DATA
 2  and GET the page indexed by <1> in folder SPEECH LOAD DATA
 3  and COPY from SPEECH ADDRESS to HERE
 4  and PASTE the "S=" in front of HERE
 5  and PASTE the FRONT in front of ":LINK #2F50"
 6  and PASTE the FRONT in front of ":REM SPEAK"
 7  and PASTE the FRONT in front of <1>
 8  and Z COMPILE the BASIC statement FRONT which means "SPEAK A PHRASE"
 9  and END   (Task is 9% full.)

SPIN AROUND CLOCKWISE this many times: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by 5.53
 3  and    SPLIT the number PRODUCT
 4  and    TAKE this many characters 4 from FRACTION
 5  and    PASTE the INTEGER in front of FRONT
 6  and    GO CLOCKWISE
 7  and    WAIT this many seconds FRONT
 8  and    STOP ALL MOTION
 9  and    END of test
10  and END   (Task is 5% full.)

SPIN AROUND COUNTERCLOCKWISE this many times: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by 5.53
 3  and    SPLIT the number PRODUCT
 4  and    TAKE this many characters 4 from FRACTION
 5  and    PASTE the INTEGER in front of FRONT
 6  and    GO COUNTERCLOCKWISE
 7  and    WAIT this many seconds FRONT
 8  and    STOP ALL MOTION
 9  and    END of test
10  and END   (Task is 5% full.)

SPIN CLOCKWISE this many degrees: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by .01472
 3  and    SPLIT the number PRODUCT
 4  and    TAKE this many characters 4 from FRACTION
 5  and    PASTE the INTEGER in front of FRONT
 6  and    GO CLOCKWISE
 7  and    WAIT this many seconds FRONT
 8  and    STOP ALL MOTION
 9  and    END of test
10  and END   (Task is 5% full.)
```

```
SPIN COUNTERCLOCKWISE this many degrees: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by .01472
 3  and    SPLIT the number PRODUCT
 4  and    TAKE this many characters 4 from FRACTION
 5  and    PASTE the INTEGER in front of FRONT
 6  and    GO COUNTERCLOCKWISE
 7  and    WAIT this many seconds FRONT
 8  and    STOP ALL MOTION
 9  and    END of test
10  and END   (Task is 5% full.)

SPIN LEFT 90 DEGREES (a Task)

 1 Does WAIT this many seconds .5
 2  and GO COUNTERCLOCKWISE
 3  and WAIT this many seconds 1.38
 4  and STOP ALL MOTION
 5  and END   (Task is 2% full.)

SPIN RIGHT 90 DEGREES (a Task)

 1 Does WAIT this many seconds .5
 2  and GO CLOCKWISE
 3  and WAIT this many seconds 1.38
 4  and STOP ALL MOTION
 5  and END   (Task is 2% full.)

STARTUP (a Task)

 1 Does CLEAR the page
 2  and DISPLAY the "Robot Control Language (RCL-II) with SAVVY -- Version 1.00
of "
 3  and DISPLAY the VERSION
 4  and CARRIAGE return
 5  and CARRIAGE return
 6  and DISPLAY the "RCL-II is Copyright (C) 1983 by RB Robot Corporation.  All
rights reserved."
 7  and CARRIAGE return
 8  and CARRIAGE return
 9  and END   (Task is 15% full.)
```

```
STOP ALL MOTION (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=0" which means "STOP ALL MOTION"
 2  and END   (Task is 3% full.)

TEST IF the variable <1> is (=,<,>,<>) <2> compared to <3>  (a Function)

 1 Does COPY from "TEST IF" to TASK NAME
 2  and Z ERROR CHECK IF/UNTIL STATEMENT
 3  and PASTE the "IF " in front of <1>
 4  and PASTE the FRONT in front of <2>
 5  and PASTE the FRONT in front of <3>
 6  and PASTE the FRONT in front of " GOTO "
 7  and Z COMPILE the BASIC statement FRONT which means "MAKE A COMPARISON"
 8  and COPY from "IF" to STRUCTURE TYPE
 9  and SAVE new page in LOOPS
10  and COPY from EMPTY to TASK NAME
11  and SAVE new page in STRUCTURES
12  and END   (Task is 9% full.)

TURN OFF LED number <1>  (a Function)

 1 Does IF the <1>  IS  1 then
 2   Do    Z COMPILE the BASIC statement "X=#FD" which means "TURN OFF LED 1"
 3  and OTHERWISE
 4   Do    IF the <1>  IS  2 then
 5   Do       Z COMPILE the BASIC statement "X=#FB" which means "TURN OFF LED

 6  and    OTHERWISE
 7   Do       IF the <1>  IS  3 then
 8   Do          Z COMPILE the BASIC statement "X=#F7" which means "TURN OFF
 3"
 9  and       OTHERWISE
10   Do          IF the <1>  IS  4 then
11   Do             Z COMPILE the BASIC statement "X=#EF" which means "TURN
LED 4"
12  and          OTHERWISE
13   Do             Z COMPILE the BASIC statement "X=#DF" which means "TURN
LED 5"
14  and             END of test
15  and          END of test
16  and       END of test
17  and    END of test
18  and Z COMPILE the BASIC statement "GOSUB 3120" which means " GO TURN OFF A
IT"
19  and Z INSURE PORT 7801 SUBROUTINE IS
20  and END   (Task is 23% full.)
```

```
TURN OFF THE FLASHING LIGHTS (a Task)

 1 Does Z COMPILE the BASIC statement "X=#BF" which means "TURN OFF FLASHING L
HTS"
 2  and Z COMPILE the BASIC statement "GOSUB 3120" which means " GO TURN OFF A
IT"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 7% full.)

TURN OFF THE HORN (a Task)

 1 Does Z COMPILE the BASIC statement "X=#7F" which means "TURN OFF HORN"
 2  and Z COMPILE the BASIC statement "GOSUB 3120" which means "GO TURN OFF A
T"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 6% full.)

TURN OFF THE INFRARED LED (a Task)

 1 Does TURN OFF LED number 1
 2  and END   (Task is 1% full.)

TURN ON LED number <1>  (a Function)

 1 Does IF the <1>  IS  1 then
 2   Do    Z COMPILE the BASIC statement "X=#02" which means "TURN ON LED 1"
 3  and OTHERWISE
 4   Do    IF the <1>  IS  2 then
 5   Do       Z COMPILE the BASIC statement "X=#04" which means "TURN ON LED 2"
 6  and    OTHERWISE
 7   Do       IF the <1>  IS  3 then
 8   Do          Z COMPILE the BASIC statement "X=#08" which means "TURN ON LEI
3"
 9  and       OTHERWISE
10   Do          IF the <1>  IS  4 then
11   Do             Z COMPILE the BASIC statement "X=#10" which means "TURN ON
ED 4"
12  and          OTHERWISE
13   Do             Z COMPILE the BASIC statement "X=#20" which means "TURN ON
ED 5"
14  and             END of test
15  and          END of test
16  and       END of test
17  and    END of test
18  and Z COMPILE the BASIC statement "GOSUB 3100" which means " GO TURN ON A B
T"
19  and Z INSURE PORT 7801 SUBROUTINE IS
20  and END   (Task is 23% full.)
```

```
TURN ON THE FLASHING LIGHTS (a Task)

 1 Does Z COMPILE the BASIC statement "X=#40" which means "TURN ON FLASHING
TS"
 2  and Z COMPILE the BASIC statement "GOSUB 3100" which means " GO TURN ON A
T"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 7% full.)

TURN ON THE HORN (a Task)

 1 Does Z COMPILE the BASIC statement "X=#80" which means "TURN ON HORN"
 2  and Z COMPILE the BASIC statement "GOSUB 3100" which means "GO TURN ON A
"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 6% full.)

TURN ON THE INFRARED LED (a Task)

 1 Does TURN ON LED number 1
 2  and END   (Task is 1% full.)

TURN THE WRIST (CW, CCW): <1> for this many degrees <2>  (a Function)

 1 Does IF the <1>  IS  CW then
 2   Do    COPY from "D=#7805:M=128:N=0:" to TEMP
 3  and OTHERWISE
 4   Do    IF the <1>  IS  CCW then
 5   Do       COPY from "D=#7805:M=192:N=64:" to TEMP
 6  and       END of test
 7  and    END of test
 8  and IF the <2>  IS GREATER THAN OR EQUAL TO  1 then
 9   Do    IF the <2>  IS LESS THAN OR EQUAL TO  360 then
10   Do       MULTIPLY the <2> by 19.88
11  and       SPLIT the number PRODUCT
12  and       COPY from INTEGER to PRODUCT
13  and       PASTE the "P=" in front of PRODUCT
14  and       PASTE the TEMP in front of FRONT
15  and       Z COMPILE the BASIC statement FRONT which means "SET THE MOTOR
C."
16  and       Z COMPILE the BASIC statement "GOSUB 3400" which means "TURN THE
RIST"
17  and       Z INSURE ARM PULSE SUBROUTINE IS
18  and       GO to the line called "SKIP"
19  and       END of test
20  and    END of test
21  and CARRIAGE return
22  and OUTPUT from "Wrist Error:  degrees = " using format DEFAULT
23  and OUTPUT from <2> using format DEFAULT
24  and HALT and return to Ready
25  and LABEL this line "SKIP"
26  and END   (Task is 24% full.)
```

```
WAIT this many seconds <1>  (a Function)

 1 Does IF the <1>  IS NOT  0 then
 2   Do    IF the <1>  IS LESS THAN OR EQUAL TO  1 then
 3   Do       MULTIPLY the <1> by 1,000
 4  and       PASTE the "DELAY " in front of PRODUCT
 5  and       Z COMPILE the BASIC statement FRONT which means "SHORT WAIT"
 6  and    OTHERWISE
 7   Do       SPLIT the number <1>
 8  and       PASTE the "T=" in front of INTEGER
 9  and       Z COMPILE the BASIC statement FRONT which means "NUMBER OF WHOLE
ECONDS"
10  and       Z COMPILE the BASIC statement "GOSUB 3000" which means "GO WAIT"
11  and       Z INSURE WAIT SUBROUTINE IS
12  and       IF the FRACTION  IS NOT  0 then
13   Do          MULTIPLY the FRACTION by 1,000
14  and          TAKE this many characters 3 from PRODUCT
15  and          PASTE the "DELAY " in front of FRONT
16  and          Z COMPILE the BASIC statement FRONT which means "MILLISECONDS
0 WAIT"
17  and          END of test
18  and       END of test
19  and    END of test
20  and END   (Task is 21% full.)

WAIT RANDOMLY up to this many seconds <1>  (a Function)

 1 Does SPLIT the number <1>
 2  and PASTE the "W=" in front of INTEGER
 3  and Z COMPILE the BASIC statement FRONT which means "MAXIMUM SECONDS"
 4  and Z COMPILE the BASIC statement "GOSUB 3070" which means "GO WAIT RANDOM
"
 5  and Z INSURE RANDOM WAIT SUBROUTINE
 6  and END   (Task is 7% full.)

X ALPHA (a Task)

 1 Does PREPARE THE ROBOT
 2  and BEGIN A LOOP
 3  and PREPARE THE ROBOT
 4  and TURN ON THE FLASHING LIGHTS
 5  and BEGIN A LOOP
 6  and TURN ON THE FLASHING LIGHTS
 7  and MOVE FORWARD
 8  and EXIT IF ANY BUMPER TOUCHED
 9  and REPEAT THIS LOOP
10  and PREPARE THE ROBOT
11  and STOP ALL MOTION
12  and MOVE BACKWARD
13  and HONK the horn for this many seconds .5
14  and WAIT this many seconds 1
15  and STOP ALL MOTION
16  and MOVE RANDOMLY
17  and WAIT RANDOMLY up to this many seconds 5
18  and STOP ALL MOTION
19  and PREPARE THE ROBOT
20  and CLEAR ALL ITEMS
21  and REPEAT THIS LOOP
22  and END   (Task is 7% full.)
```

```
TASKS
Name        (Status)

 ASSIGN A MOTOR CODE
*BEGIN A COUNTED LOOP called <1> beginning at <2> ending at <3>
*BEGIN A LIMITED LOOP
*BEGIN A LOOP
*BUILD a program from the robot task <1>
*BUILD AND LOAD
*CALCULATE variable <1> = <2> (+,-,*,/) <3>  <4>
*CALL this robot task line <1>
 CLEAR ALL ITEMS
*DEFINE A VARIABLE called <1>
 END HERE
 END OF PROGRAM
*EXIT IF ANY BUMPER TOUCHED
*EXIT IF FRONT+REAR BUMPER PRESS
*EXIT IF SONAR distance value is less than <1>
*EXIT IF THE BATTERY IS LOW
*EXIT IF THE CHARGER IS TOUCHED
*EXIT IF THE TAPE IS SENSED
*EXIT IF THIS BUMPER is touched <1>
*EXIT THIS LOOP
*FOLLOW TAPE
*GO CLOCKWISE
*GO COUNTERCLOCKWISE
*HONK the horn for this many seconds <1>
 INITIALIZE MEMORY
 INITIALIZE VARIABLES
*JUMP to the line called <1>
*LIGHTS ROUTINE
*LIST THE PROGRAM
*LOAD
*LOAD THE PHRASE called: <1> with phonemes: <2>
*MAINTAIN CHARGE
*MOVE BACKWARD
*MOVE DISTANCE BACKWARD for this many feet: <1>
*MOVE DISTANCE FORWARD for this many feet: <1>
*MOVE FORWARD
*MOVE FORWARD TIL TAPE NOT SENSED
*MOVE RANDOMLY
*MOVE THE ARM FROM SHOULDER (UP,DOWN,IN,OUT): <1> for this many degrees <2>
*MOVE THE FOREARM (IN,OUT): <1> for this many degrees <2>
*MOVE THE HAND (OPEN,CLOSE): <1> for this (1-100) %: <2>
*MOVE TIMED BACKWARD for this many seconds <1>
*MOVE TIMED FORWARD for this many seconds <1>
*MOVE WITH BETA INTELLIGENCE
 OTHERWISE DO
*PICK A RANDOM DIRECTION
*PIVOT ON LEFT CLOCKWISE
*PIVOT ON LEFT COUNTERCLOCKWISE
*PIVOT ON RIGHT CLOCKWISE
*PIVOT ON RIGHT COUNTERCLOCKWISE
*PREPARE THE ROBOT
*PREPARE THE VOICE
 REMARK that: <1>
*REPEAT THE LIMITED LOOP unless <1> is (<,>,=,<>) <2> to <3>
*REPEAT THIS COUNTED LOOP for the counter <1>
*REPEAT THIS LOOP
 REPORT ASSIGN ERROR
```

```
TASKS
Name        (Status)

 REPORT DELETE ERROR
 REPORT GET ERROR  (Undefined)
 REPORT LOAD ERROR
 REPORT MATH ERROR
 REPORT REPLACE ERROR
 REPORT SAVE ERROR
*RUN
*SET the variable <1> equal to <2>
 SHOW ERRORS
 SHOW THE PHONEME DICTIONARY
*SPEAK the phrase called:  <1>
*SPIN AROUND CLOCKWISE this many times: <1>
*SPIN AROUND COUNTERCLOCKWISE this many times: <1>
*SPIN CLOCKWISE this many degrees: <1>
*SPIN COUNTERCLOCKWISE this many degrees: <1>
*SPIN LEFT 90 DEGREES
*SPIN RIGHT 90 DEGREES
 STARTUP
*STOP ALL MOTION
*TEST IF the variable <1> is (=,<,>,<>) <2> compared to <3>
*TURN OFF LED number <1>
*TURN OFF THE FLASHING LIGHTS
*TURN OFF THE HORN
*TURN OFF THE INFRARED LED
*TURN ON LED number <1>
*TURN ON THE FLASHING LIGHTS
*TURN ON THE HORN
*TURN ON THE INFRARED LED
*TURN THE WRIST (CW, CCW): <1> for this many degrees <2>
*WAIT this many seconds <1>
*WAIT RANDOMLY up to this many seconds <1>
*X ALPHA
*X ALPHA W/SONAR
*X BETA
*X BETA W/SONAR
*X BUMPER ACTIVATED MOTIONS
*X CHARGER FINDER
*X LOAD THE INTRODUCTION
*X SIMPLE SIMON
 Z COMPILE the BASIC statement <1> which means <2>
 Z COMPUTE 2 TO THE N-1 , with N = <1>
 Z DELAY for this many loops:  <1>
 Z DO CHECK if item <1> is in folder <2>
 Z DO NUMBER TEST on <1>
 Z EMPTY THE PROGRAM
 Z ERROR CHECK IF/UNTIL STATEMENT
 Z ERROR CHECK MATH FUNCTION
 Z ERROR CHECK STRUCTURES
 Z ERROR CHECK THE FOR STATEMENT
 Z ERROR CHECKING?
 Z FIND the label called <1>
 Z GET PREVIOUS BEGIN
 Z GET PREVIOUS IF OR ELSE
 Z INCLUDE at <1> the statement <2> which means <3>
 Z INCREMENT the <1>
 Z INCREMENT STATEMENT NUMBER
 Z INITIALIZE LEGAL VARIABLES
```

```
TASKS
Name        (Status)

Z INSURE ARM PULSE SUBROUTINE IS
Z INSURE BETA SUBROUTINE IS
Z INSURE END OF PROGRAM IS THERE
Z INSURE PORT 7801 SUBROUTINE IS
Z INSURE RANDOM TURN SUBROUTINE
Z INSURE RANDOM WAIT SUBROUTINE
Z INSURE VOICE SUBROUTINE IS
Z INSURE WAIT SUBROUTINE IS
Z LOAD SONAR
Z LOAD VOICE SUBROUTINE
Z LOOK FOR BEGIN
Z LOOK FOR MATCHING STATEMENT
Z OUTPUT PAGE NUMBER
Z OUTPUT PROGRAM HEADING
Z PATCH UP END STATEMENTS
Z PATCH UP EXIT STATEMENTS
Z PATCH UP JUMP STATEMENTS
Z SAVE FOLDER ERROR
Z SAVE NUMBER OR FOLDER ERROR
Z SAVE STRUCTURE ERROR missing the statement <1> to go with the statment <2>
statement number <3>
ZZ AVAIL TASK
ZZ AVAIL TASK #1
ZZ AVAIL TASK #10
ZZ AVAIL TASK #11
ZZ AVAIL TASK #12
ZZ AVAIL TASK #13
ZZ AVAIL TASK #14
ZZ AVAIL TASK #2
ZZ AVAIL TASK #3
ZZ AVAIL TASK #4
ZZ AVAIL TASK #5
ZZ AVAIL TASK #6
ZZ AVAIL TASK #7
ZZ AVAIL TASK #8
```

APPENDIX 2

Detail of Provided RCL Tasks

```
ASSIGN A MOTOR CODE (a Task)

 1 Does Z COMPILE the BASIC statement "IF Y=251 THEN P=9" which means "FORWAF
 2  and Z COMPILE the BASIC statement "IF Y=254 THEN P=8" which means "RIGHT _C
WARD"
 3  and Z COMPILE the BASIC statement "IF Y=247 THEN P=5" which means "ROTATE F
GHT"
 4  and Z COMPILE the BASIC statement "IF Y=127 THEN P=7" which means "RIGHT RE
ERSE"
 5  and Z COMPILE the BASIC statement "IF Y=239 THEN P=6" which means "REVERS
 6  and Z COMPILE the BASIC statement "IF Y=191 THEN P=2" which means "LEFT R_V
RSE"
 7  and Z COMPILE the BASIC statement "IF Y=223 THEN P=10" which means "ROTATE
EFT"
 8  and Z COMPILE the BASIC statement "IF Y=253 THEN P=1" which means "LEFT FOR
ARD"
 9  and Z COMPILE the BASIC statement "@M=P" which means "ASSIGN THE MOTOR CC E

10  and END   (Task is 32% full.)

BEGIN A COUNTED LOOP called <1> beginning at <2> ending at <3>  (a Function)

 1 Does Z ERROR CHECK THE FOR STATEMENT
 2  and PASTE the "FOR " in front of <1>
 3  and PASTE the FRONT in front of "="
 4  and PASTE the FRONT in front of <2>
 5  and PASTE the FRONT in front of " TO "
 6  and PASTE the FRONT in front of <3>
 7  and Z COMPILE the BASIC statement FRONT which means "BEGIN A COUNTED LOOP
OR/NEXT""
 8  and COPY from "FOR" to STRUCTURE TYPE
 9  and SAVE new page in STRUCTURES
10  and END   (Task is 9% full.)

BEGIN A LIMITED LOOP (a Task)

 1 Does Z COMPILE the BASIC statement "DO" which means "BEGIN A LIMITED LOOP"
 2  and COPY from "DO" to STRUCTURE TYPE
 3  and SAVE new page in STRUCTURES
 4  and END   (Task is 4% full.)

BEGIN A LOOP (a Task)

 1 Does Z COMPILE the BASIC statement "REM START A LOOP" which means "BEGIN A L
OP"
 2  and COPY from "BEGIN" to STRUCTURE TYPE
 3  and SAVE new page in LOOPS
 4  and SAVE new page in STRUCTURES
 5  and END   (Task is 5% full.)
```

```
BUILD a program from the robot task <1>  (a Function)

 1 Does CLEAR the page
 2  and COPY from "BUILD" to ASSIGNMENT
 3  and ASSIGN the <1> to ASSIGNED TASK
 4  and Z ERROR CHECKING?
 5  and CLEAR the page
 6  and Z EMPTY THE PROGRAM
 7  and DISPLAY the "Now building robot task "
 8  and DISPLAY the <1>
 9  and DISPLAY the "...one moment, please."
10  and CARRIAGE return
11  and DO the assigned task
12  and Z PATCH UP JUMP STATEMENTS
13  and Z INSURE END OF PROGRAM IS THERE
14  and COPY from EMPTY to ASSIGNMENT
15  and IF the ERROR CHECK  IS  YES then
16   Do    COPY from 199 to ERROR NUMBER
17  and    Z ERROR CHECK STRUCTURES
18  and    GET the page indexed by FIRST in folder ERRORS
19  and    IF the PAGE  IS NOT  EMPTY then
20   Do       SHOW ERRORS
21  and       END of test
22  and    END of test
23  and END   (Task is 13% full.)

BUILD AND LOAD (a Task)

 1 Does CLEAR the page
 2  and OUTPUT from "*** Build and load an RBSX robot task ***" using format .
AULT
 3  and CARRIAGE return
 4  and CARRIAGE return
 5  and OUTPUT from "Please enter the name of the robot task:  " using format
FAULT
 6  and INPUT into HERE
 7  and BUILD a program from the robot task HERE
 8  and CLEAR the page
 9  and OUTPUT from "** Build completed, now loading into the RBSX **" using f
mat DEFAULT
10  and LOAD
11  and END   (Task is 16% full.)
```

```
CALCULATE variable <1> = <2> (+,-,*,/) <3>  <4>  (a Function)

 1 Does Z ERROR CHECK MATH FUNCTION
 2  and PASTE the <1> in front of "="
 3  and PASTE the FRONT in front of <2>
 4  and PASTE the FRONT in front of <3>
 5  and PASTE the FRONT in front of <4>
 6  and Z COMPILE the BASIC statement FRONT which means "MATH FUNCTION"
 7  and END   (Task is 5% full.)

CALL this robot task line <1>  (a Function)

 1 Does PASTE the "REM LABEL THIS LINE " in front of <1>
 2  and Z COMPILE the BASIC statement FRONT which means "LABEL THE LINE"
 3  and PASTE the "L" in front of <1>
 4  and COPY from FRONT to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and END   (Task is 6% full.)

CLEAR ALL ITEMS (a Task)

 1 Does Z COMPILE the BASIC statement "CLEAR" which means "CLEAR VARIABLES"
 2  and END   (Task is 3% full.)

DEFINE A VARIABLE called <1>  (a Function)

 1 Does BEGIN at page indexed by FIRST in folder LEGAL VARIABLES
 2  and START a loop
 3   Do    GET the page indexed by NEXT in folder LEGAL VARIABLES
 4  and    IF the USED  IS  NO then
 5   Do       LEAVE this loop
 6  and       END of test
 7  and    REPEAT this loop
 8  and COPY from <1> to VARIABLES ID
 9  and COPY from LEGAL ID to VAR NAME
10  and SAVE new page in VARIABLES
11  and COPY from YES to USED
12  and REPLACE this page in folder LEGAL VARIABLES
13  and END   (Task is 5% full.)

END HERE (a Task)

 1 Does Z COMPILE the BASIC statement "REM END HERE" which means "FALL THROU
O HERE"
 2  and Z PATCH UP END STATEMENTS
 3  and COPY from "END" to STRUCTURE TYPE
 4  and SAVE new page in STRUCTURES
 5  and END   (Task is 5% full.)
```

```
END OF PROGRAM (a Task)

 1 Does Z INCLUDE at 2,999 the statement "GOTO 3400" which means "GOTO END"
 2  and Z INCLUDE at 3,600 the statement "REM END OF PROGRAM" which means ""
 3  and END   (Task is 6% full.)

EXIT IF ANY BUMPER TOUCHED (a Task)

 1 Does Z COMPILE the BASIC statement "Y=@#7800" which means "TEST FOR BUMPER
NTACT"
 2  and Z COMPILE the BASIC statement "IF Y<255 GOTO EXIT" which means "EXIT I
ANY CONTACT"
 3  and COPY from "EXIT" to STRUCTURE TYPE
 4  and SAVE new page in LOOPS
 5  and SAVE new page in STRUCTURES
 6  and END   (Task is 10% full.)

EXIT IF FRONT+REAR BUMPER PRESS (a Task)

 1 Does Z COMPILE the BASIC statement "Y=@#7800" which means "TEST FOR BUMPER
NTACT"
 2  and Z COMPILE the BASIC statement "IF Y=235 GOTO EXIT" which means "EXIT I
FRONT+REAR BUMPER PRESS"
 3  and COPY from "EXIT" to STRUCTURE TYPE
 4  and SAVE new page in LOOPS
 5  and SAVE new page in STRUCTURES
 6  and END   (Task is 11% full.)

EXIT IF SONAR distance value is less than <1>  (a Function)

 1 Does Z COMPILE the BASIC statement "D=0" which means "ZERO DISTANCE"
 2  and Z COMPILE the BASIC statement "LINK #1800" which means "CALL SONAR"
 3  and SPLIT the number <1>
 4  and PASTE the "IF D<" in front of INTEGER
 5  and PASTE the FRONT in front of " GOTO EXIT"
 6  and Z COMPILE the BASIC statement FRONT which means "EXIT IF LESS THAN VALU
"
 7  and COPY from "EXIT" to STRUCTURE TYPE
 8  and SAVE new page in LOOPS
 9  and COPY from YES to SONAR NEEDED
10  and SAVE new page in STRUCTURES
11  and END   (Task is 14% full.)
```

```
EXIT IF THE BATTERY IS LOW (a Task)

 1 Does Z COMPILE the BASIC statement "Z=@#7802 AND #10" which means "TEST I
TTERY LOW"
 2  and Z COMPILE the BASIC statement "IF Z=0 GOTO EXIT" which means "EXIT IF
TTERY IS LOW"
 3  and COPY from "EXIT" to STRUCTURE TYPE
 4  and SAVE new page in LOOPS
 5  and SAVE new page in STRUCTURES
 6  and END   (Task is 10% full.)

EXIT IF THE CHARGER IS TOUCHED (a Task)

 1 Does Z COMPILE the BASIC statement "Q=@#7802 AND #20" which means "TEST F R
HARGER CONTACT"
 2  and Z COMPILE the BASIC statement "IF Q=0 GOTO EXIT" which means "EXIT IF
ARGER IS TOUCHED"
 3  and COPY from "EXIT" to STRUCTURE TYPE
 4  and SAVE new page in LOOPS
 5  and SAVE new page in STRUCTURES
 6  and END   (Task is 11% full.)

EXIT IF THE TAPE IS SENSED (a Task)

 1 Does TURN ON THE INFRARED LED
 2  and Z COMPILE the BASIC statement "R=@#7802 AND #40" which means "TEST FOR
APE SENSE"
 3  and Z COMPILE the BASIC statement "IF R=0 GOTO EXIT" which means "EXIT I
PE SENSED"
 4  and COPY from "EXIT" to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and SAVE new page in STRUCTURES
 7  and END   (Task is 10% full.)

EXIT IF THIS BUMPER is touched <1>  (a Function)

 1 Does IF the <1>  IS LESS THAN OR EQUAL TO  8 then
 2   Do    IF the <1>  IS GREATER THAN OR EQUAL TO  1 then
 3  Do       Z COMPILE the BASIC statement "Y=@#7800" which means "TEST FOR B
PER CONTACT"
 4  and      SPLIT the number <1>
 5  and      Z COMPUTE 2 TO THE N-1 , with N = INTEGER
 6  and      SUBTRACT the PRODUCT from 255
 7  and      PASTE the "IF Y=" in front of DIFFERENCE
 8  and      PASTE the FRONT in front of " GOTO EXIT"
 9  and      Z COMPILE the BASIC statement FRONT which means "EXIT IF CONTA T
10  and      COPY from "EXIT" to STRUCTURE TYPE
11  and      SAVE new page in LOOPS
12  and      SAVE new page in STRUCTURES
13  and      END of test
14  and   END of test
15  and END   (Task is 15% full.)
```

```
EXIT THIS LOOP (a Task)

 1 Does Z COMPILE the BASIC statement "GOTO GOES HERE" which means "EXIT THIS
OP"
 2  and COPY from "EXIT" to STRUCTURE TYPE
 3  and SAVE new page in LOOPS
 4  and SAVE new page in STRUCTURES
 5  and END    (Task is 5% full.)

FOLLOW TAPE (a Task)

 1 Does BEGIN A LOOP
 2  and PREPARE THE ROBOT
 3  and CLEAR ALL ITEMS
 4  and BEGIN A LOOP
 5  and PIVOT ON LEFT COUNTERCLOCKWISE
 6  and EXIT IF THE TAPE IS SENSED
 7  and EXIT IF THE CHARGER IS TOUCHED
 8  and REPEAT THIS LOOP
 9  and EXIT IF THE CHARGER IS TOUCHED
10  and TURN ON THE FLASHING LIGHTS
11  and MOVE FORWARD TIL TAPE NOT SENSED
12  and PREPARE THE ROBOT
13  and CLEAR ALL ITEMS
14  and BEGIN A LOOP
15  and PIVOT ON RIGHT CLOCKWISE
16  and EXIT IF THE TAPE IS SENSED
17  and EXIT IF THE CHARGER IS TOUCHED
18  and REPEAT THIS LOOP
19  and EXIT IF THE CHARGER IS TOUCHED
20  and TURN ON THE FLASHING LIGHTS
21  and MOVE FORWARD TIL TAPE NOT SENSED
22  and REPEAT THIS LOOP
23  and END    (Task is 6% full.)

60 CLOCKWISE (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#05" which means "SPIN CLOCKWISE
 2  and END    (Task is 3% full.)

60 COUNTERCLOCKWISE (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#0A" which means "SPIN COUNTERCL
KWISE"
 2  and END    (Task is 4% full.)
```

```
HONK the horn for this many seconds <1>  (a Function)

 1 Does TURN ON THE HORN
 2  and WAIT this many seconds <1>
 3  and TURN OFF THE HORN
 4  and END   (Task is 1% full.)

INITIALIZE MEMORY (a Task)

 1 Does Z COMPILE the BASIC statement "N=TOP" which means "INITIALIZE EXPER
 BLOCK"
 2  and Z COMPILE the BASIC statement "O=TOP+#FF" which means "INITIALIZE IN
TION BLOCK"
 3  and Z COMPILE the BASIC statement "M=TOP+#200" which means ""
 4  and Z COMPILE the BASIC statement "FOR P=N TO M" which means ""
 5  and Z COMPILE the BASIC statement "@P=#FF" which means ""
 6  and Z COMPILE the BASIC statement "NEXT P" which means ""
 7  and END   (Task is 15% full.)

INITIALIZE VARIABLES (a Task)

 1 Does Z COMPILE the BASIC statement "N=TOP" which means ""
 2  and Z COMPILE the BASIC statement "O=TOP+#FF" which means ""
 3  and Z COMPILE the BASIC statement "M=N" which means ""
 4  and Z COMPILE the BASIC statement "FOR P=N TO O" which means "INITIALIZE
ORY"
 5  and Z COMPILE the BASIC statement "@P=0" which means ""
 6  and Z COMPILE the BASIC statement "NEXT P" which means ""
 7  and END   (Task is 11% full.)

JUMP to the line called <1>  (a Function)

 1 Does PASTE the "JUMP TO LABEL " in front of <1>
 2  and Z COMPILE the BASIC statement "GOTO GOES HERE" which means FRONT
 3  and PASTE the "J" in front of <1>
 4  and COPY from FRONT to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and END   (Task is 6% full.)

LIGHTS ROUTINE (a Task)

 1 Does TURN ON LED number 2
 2  and WAIT this many seconds .1
 3  and TURN ON LED number 3
 4  and WAIT this many seconds .1
 5  and TURN ON LED number 4
 6  and WAIT this many seconds .1
 7  and TURN ON LED number 5
 8  and WAIT this many seconds .1
 9  and TURN ON THE FLASHING LIGHTS
10  and WAIT this many seconds 1
11  and END   (Task is 7% full.)
```

LIST THE PROGRAM (a Task)

```
 1 Does BEGIN at page indexed by FIRST in folder SOURCE
 2  and COPY from 0 to PAGE COUNT
 3  and START a loop
 4   Do    CLEAR the page
 5  and    Z INCREMENT the PAGE COUNT
 6  and    Z OUTPUT PAGE NUMBER
 7  and    Z OUTPUT PROGRAM HEADING
 8  and    COPY from 5 to LINE COUNT
 9  and    START a loop
10   Do       Z INCREMENT the LINE COUNT
11  and       GET the page indexed by NEXT in folder SOURCE
12  and       IF the PAGE  IS  EMPTY then
13   Do          LEAVE this loop
14  and          END of test
15  and       OUTPUT from SPACES using format 5
16  and       OUTPUT from STATEMENT NUMBER using format "#4"
17  and       OUTPUT from SPACES using format 5
18  and       OUTPUT from BASIC TEXT using format 30
19  and       OUTPUT from SPACES using format 5
20  and       OUTPUT from MEANING using format 30
21  and       CARRIAGE return
22  and       IF the LINE COUNT  IS  55 then
23   Do          LEAVE this loop
24  and          END of test
25  and       REPEAT this loop
26  and    IF the PAGE  IS  EMPTY then
27   Do       LEAVE this loop
28  and       END of test
29  and    REPEAT this loop
30  and CARRIAGE return
31  and CARRIAGE return
32  and OUTPUT from SPACES using format 26
33  and OUTPUT from "** End of the Program **" using format DEFAULT
34  and CARRIAGE return
35  and END   (Task is 19% full.)
```

LOAD (a Task)

```
 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  10 then
 2   Do    REPORT LOAD ERROR
 3  and    END of test
 4  and CLEAR the page
 5  and RING bell
 6  and RING bell
 7  and RING bell
 8  and DISPLAY the "Ready to load into the RB5X...press any key to continue."
 9  and LIMIT the input for ANSWER to a maximum of 1
10  and CLEAR the page
11  and DISPLAY the "** Now loading...One moment please **"
12  and CARRIAGE return
13  and CARRIAGE return
14  and DISPLAY the "Now loading your robot task..."
15  and CARRIAGE return
16  and PRINT the ""
17  and OUTPUT from "@#7803=#98:@#7801=#7C" using format DEFAULT
18  and CARRIAGE return
19  and Z DELAY for this many loops:  10
20  and OUTPUT from "NEW #1000" using format DEFAULT
```

```
LOAD (a Task)

21  and CARRIAGE return
22  and Z DELAY for this many loops:  60
23  and OUTPUT from "NEW" using format DEFAULT
24  and CARRIAGE return
25  and Z DELAY for this many loops:  5
26  and BEGIN at page indexed by FIRST in folder SOURCE
27  and IF the VOICE NEEDED  IS  YES then
28   Do    Z LOAD VOICE SUBROUTINE
29  and    DISPLAY the ""
30  and    CARRIAGE return
31  and    PRINT the ""
32  and    END of test
33  and IF the SONAR NEEDED  IS  YES then
34   Do    Z LOAD SONAR
35  and    DISPLAY the ""
36  and    CARRIAGE return
37  and    PRINT the ""
38  and    END of test
39  and DISPLAY the ""
40  and CARRIAGE return
41  and DISPLAY the "Now loading your Tiny BASIC robot task..."
42  and CARRIAGE return
43  and PRINT the ""
44  and START a loop
45   Do    GET the page indexed by NEXT in folder SOURCE
46  and    IF the PAGE  IS  EMPTY then
47   Do       LEAVE this loop
48  and       END of test
49  and    PASTE the STATEMENT NUMBER in front of " "
50  and    PASTE the FRONT in front of BASIC TEXT
51  and    OUTPUT from FRONT using format DEFAULT
52  and    CARRIAGE return
53  and    DISPLAY the "#"
54  and    PRINT the ""
55  and    Z DELAY for this many loops:  5
56  and    REPEAT this loop
57  and Z DELAY for this many loops:  15
58  and PRINT the ""
59  and OUTPUT from "RUN" using format DEFAULT
60  and CARRIAGE return
61  and DISPLAY the ""
62  and RING bell
63  and CLEAR the page
64  and DISPLAY the "*** Your robot task has been successfully loaded ***"
65  and END   (Task is 47% full.)
```

```
LOAD THE PHRASE called: <1> with phonemes: <2>  (a Function)

 1 Does COPY from 0 to SPEECH LENGTH
 2  and COPY from EMPTY to SPEECH ADDRESS
 3  and GET the page indexed by FIRST in folder SPEECH LOAD DATA
 4  and DISPLAY the PAGE
 5  and IF the PAGE  IS  EMPTY then
 6   Do    COPY from "TOP+500" to SPEECH ADDRESS
 7  and    COPY from "TEMP" to LABEL ID
 8  and    SAVE new page in SPEECH LOAD DATA
 9  and    GET the page indexed by "TEMP" in folder SPEECH LOAD DATA
10  and    END of test
11  and GET the page indexed by "TEMP" in folder SPEECH LOAD DATA
12  and Z INSURE VOICE SUBROUTINE IS
13  and START a loop
14   Do    IF the <2>  IS  EMPTY then
15   Do       LEAVE this loop
16  and       END of test
17  and    ADD the 1 and SPEECH LENGTH
18  and    COPY from SUM to SPEECH LENGTH
19  and    CUT the <2> after DELIMITER
20  and    COPY from BACK to <2>
21  and    SUBTRACT the 1 from LENGTH
22  and    TAKE this many characters DIFFERENCE from FRONT
23  and    WHEN folder PHONEMES  CONTAINS  FRONT then
24   Do       GET the page indexed by FRONT in folder PHONEMES
25  and       PASTE the "P=#" in front of PHONEME CODE
26  and       PASTE the FRONT in front of ":GOSUB 3500"
27  and       Z COMPILE the BASIC statement FRONT which means "VOICE SUBROUTIN

28  and       COPY from YES to VOICE NEEDED
29  and    OTHERWISE
30   Do       IF the ERROR CHECK  IS  YES then
31   Do          Z COMPILE the BASIC statement "REM PHONEME ERROR" which means
"
32  and          Z INCREMENT the ERROR NUMBER
33  and          COPY from STATEMENT NUMBER to LINE NUMBER
34  and          PASTE the PHONEME ID in front of " is not a valid phoneme"
35  and          COPY from FRONT to ERROR MESSAGE
36  and          SAVE new page in ERRORS
37  and          END of test
38  and       END of test
39  and    REPEAT this loop
40  and GET the page indexed by LAST in folder SPEECH LOAD DATA
41  and COPY from <1> to LABEL ID
42  and SAVE new page in SPEECH LOAD DATA
43  and PASTE the SPEECH ADDRESS in front of "+"
44  and PASTE the FRONT in front of SPEECH LENGTH
45  and COPY from FRONT to SPEECH ADDRESS
46  and COPY from "TEMP" to LABEL ID
47  and COPY from SPEECH ADDRESS to HERE
48  and GET the page indexed by "TEMP" in folder SPEECH LOAD DATA
49  and COPY from HERE to SPEECH ADDRESS
50  and REPLACE this page in folder SPEECH LOAD DATA
51  and END   (Task is 38% full.)
```

```
MAINTAIN CHARGE (a Task)

 1 Does BEGIN A LOOP
 2  and BEGIN A LOOP
 3  and EXIT IF THE CHARGER IS TOUCHED
 4  and SET the variable "P" equal to 0
 5  and BEGIN A LOOP
 6  and Z COMPILE the BASIC statement "P=P+1" which means "INCREMENT LOOP C(

 7  and MOVE TIMED FORWARD for this many seconds .1
 8  and EXIT IF THE CHARGER IS TOUCHED
 9  and TEST IF the variable "P" is (=,<,>,<>) "=" compared to 5
10  and MOVE TIMED BACKWARD for this many seconds 1
11  and MOVE TIMED FORWARD for this many seconds 1.1
12  and SET the variable "P" equal to 0
13  and END HERE
14  and REPEAT THIS LOOP
15  and REPEAT THIS LOOP
16  and TURN ON LED number 2
17  and WAIT this many seconds 1
18  and PREPARE THE ROBOT
19  and REPEAT THIS LOOP
20  and END   (Task is 14% full.)

MOVE BACKWARD (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#06" which means "REVERSE"
 2  and END   (Task is 2% full.)

MOVE DISTANCE BACKWARD for this many feet: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by 3
 3  and    MOVE BACKWARD
 4  and    WAIT this many seconds PRODUCT
 5  and    STOP ALL MOTION
 6  and    END of test
 7  and END   (Task is 3% full.)

MOVE DISTANCE FORWARD for this many feet: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by 3
 3  and    MOVE FORWARD
 4  and    WAIT this many seconds PRODUCT
 5  and    STOP ALL MOTION
 6  and    END of test
 7  and END   (Task is 3% full.)
```

```
MOVE FORWARD (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#09" which means "GO FORWARD"
 2  and END   (Task is 3% full.)

MOVE FORWARD TIL TAPE NOT SENSED (a Task)

 1 Does SET the variable "T" equal to 0
 2  and Z COMPILE the BASIC statement "DO" which means "BEGIN A LIMITED LOOP"
 3  and CALCULATE variable "T" = "T" (+,-,*,/) "+"  1
 4  and MOVE FORWARD
 5  and Z COMPILE the BASIC statement "R=@#7802 AND #40" which means "TEST FOR
APE SENSE"
 6  and Z COMPILE the BASIC statement "UNTIL (R<>0) OR (T>=100)" which means "
ECK EXIT CONDITIONS"
 7  and WAIT this many seconds .1
 8  and END   (Task is 17% full.)

MOVE RANDOMLY (a Task)

 1 Does Z COMPILE the BASIC statement "GOSUB 3040" which means "PICK RANDOM DI
CTION"
 2  and Z COMPILE the BASIC statement "@#7802=V" which means "GO RANDOM DIRECT
N"
 3  and Z INSURE RANDOM TURN SUBROUTINE
 4  and END   (Task is 7% full.)

MOVE THE ARM FROM SHOULDER (UP,DOWN,IN,OUT): <1> for this many degrees <2>  (a
unction)

 1 Does IF the <1>  IS  IN then
 2   Do    COPY from "D=#7805:M=8:N=0:" to TEMP
 3  and OTHERWISE
 4   Do    IF the <1>  IS  OUT then
 5   Do       COPY from "D=#7805:M=12:N=4:" to TEMP
 6  and    OTHERWISE
 7   Do       IF the <1>  IS  DOWN then
 8   Do          COPY from "D=#7805:M=3:N=1:" to TEMP
 9  and       OTHERWISE
10   Do          IF the <1>  IS  UP then
11   Do             COPY from "D=#7805:M=2:N=0:" to TEMP
12  and             END of test
13  and          END of test
14  and       END of test
15  and    END of test
16  and IF the <2>  IS GREATER THAN OR EQUAL TO  1 then
17   Do    IF the <2>  IS LESS THAN OR EQUAL TO  360 then
18   Do       MULTIPLY the <2> by 40.38
19  and       SPLIT the number PRODUCT
20  and       COPY from INTEGER to PRODUCT
21  and       PASTE the "P=" in front of PRODUCT
22  and       PASTE the TEMP in front of FRONT
23  and       Z COMPILE the BASIC statement FRONT which means "SET MOTOR, ETC."
24  and       Z COMPILE the BASIC statement "GOSUB 3400" which means "MOVE THE
RM"
25  and       Z INSURE ARM PULSE SUBROUTINE IS
26  and       GO to the line called "SKIP"
27  and       END of test
28  and    END of test
29  and CARRIAGE return
```

```
MOVE THE ARM FROM SHOULDER (UP,DOWN,IN,OUT): <1> for this many degrees <2>
unction)

30  and OUTPUT from "Shoulder Error:  degrees = " using format DEFAULT
31  and OUTPUT from <2> using format DEFAULT
32  and HALT and return to Ready
33  and LABEL this line "SKIP"
34  and END   (Task is 30% full.)

MOVE THE FOREARM (IN,OUT): <1> for this many degrees <2>  (a Function)

 1 Does IF the <1>  IS  IN then
 2   Do    COPY from "D=#7805:M=48:N=16:" to TEMP
 3  and OTHERWISE
 4   Do    IF the <1>  IS  OUT then
 5   Do       COPY from "D=#7805:M=32:N=0:" to TEMP
 6  and       END of test
 7  and    END of test
 8  and IF the <2>  IS GREATER THAN OR EQUAL TO  1 then
 9   Do    IF the <2>  IS LESS THAN OR EQUAL TO  360 then
10   Do       MULTIPLY the <2> by 20.16
11  and       SPLIT the number PRODUCT
12  and       COPY from INTEGER to PRODUCT
13  and       PASTE the "P=" in front of PRODUCT
14  and       PASTE the TEMP in front of FRONT
15  and       Z COMPILE the BASIC statement FRONT which means "SET THE MOTOR,
C."
16  and       Z COMPILE the BASIC statement "GOSUB 3400" which means "MOVE TI
RM"
17  and       Z INSURE ARM PULSE SUBROUTINE IS
18  and       GO to the line called "SKIP"
19  and       END of test
20  and    END of test
21  and CARRIAGE return
22  and OUTPUT from "Forearm Error:  degrees = " using format DEFAULT
23  and OUTPUT from <2> using format DEFAULT
24  and HALT and return to Ready
25  and LABEL this line "SKIP"
26  and END   (Task is 24% full.)

MOVE THE HAND (OPEN,CLOSE): <1> for this (1-100) %: <2>  (a Function)

 1 Does IF the <1>  IS  OPEN then
 2   Do    COPY from "D=#7806:M=242:N=240:" to TEMP
 3  and OTHERWISE
 4   Do    IF the <1>  IS  CLOSE then
 5   Do       COPY from "D=#7806:M=243:N=241:" to TEMP
 6  and       END of test
 7  and    END of test
 8  and IF the <2>  IS GREATER THAN OR EQUAL TO  1 then
 9   Do    IF the <2>  IS LESS THAN OR EQUAL TO  100 then
10   Do       MULTIPLY the <2> by 4.42
11  and       SPLIT the number PRODUCT
12  and       COPY from INTEGER to PRODUCT
13  and       SPLIT the number PRODUCT
14  and       PASTE the "P=" in front of INTEGER
15  and       PASTE the TEMP in front of FRONT
16  and       Z COMPILE the BASIC statement FRONT which means "SET MOTORS, ETC

17  and       Z COMPILE the BASIC statement "GOSUB 3400" which means "MOVE TH
AND"
```

```
MOVE THE HAND (OPEN,CLOSE): <1> for this (1-100) %: <2>  (a Function)

18  and        Z INSURE ARM PULSE SUBROUTINE IS
19  and        GO to the line called "SKIP"
20  and        END of test
21  and     END of test
22  and CARRIAGE return
23  and OUTPUT from "Hand Error:  percent = " using format DEFAULT
24  and OUTPUT from <2> using format DEFAULT
25  and HALT and return to Ready
26  and LABEL this line "SKIP"
27  and END   (Task is 24% full.)

MOVE TIMED BACKWARD for this many seconds <1>  (a Function)

 1 Does MOVE BACKWARD
 2  and WAIT this many seconds <1>
 3  and STOP ALL MOTION
 4  and END   (Task is 1% full.)

MOVE TIMED FORWARD for this many seconds <1>  (a Function)

 1 Does MOVE FORWARD
 2  and WAIT this many seconds <1>
 3  and STOP ALL MOTION
 4  and END   (Task is 1% full.)

MOVE WITH BETA INTELLIGENCE (a Task)

 1 Does Z COMPILE the BASIC statement "GOSUB 3160" which means "GO TO BETA SU
UTINE"
 2  and Z COMPILE the BASIC statement "REM RETURN GOES HERE" which means "RETU
 TO HERE"
 3  and COPY from STATEMENT NUMBER to RETURN STATEMENT NUMBER
 4  and Z INSURE BETA SUBROUTINE IS
 5  and END   (Task is 9% full.)

OTHERWISE DO (a Task)

 1 Does Z COMPILE the BASIC statement "REM ELSE" which means "DO THIS INSTEAD"
 2  and COPY from "ELSE" to STRUCTURE TYPE
 3  and SAVE new page in LOOPS
 4  and SAVE new page in STRUCTURES
 5  and END   (Task is 5% full.)
```

```
X ALPHA W/SONAR (a Task)

 1 Does PREPARE THE ROBOT
 2  and BEGIN A LOOP
 3  and PREPARE THE ROBOT
 4  and TURN ON THE FLASHING LIGHTS
 5  and BEGIN A LOOP
 6  and MOVE FORWARD
 7  and EXIT IF SONAR distance value is less than 95
 8  and EXIT IF ANY BUMPER TOUCHED
 9  and REPEAT THIS LOOP
10  and PREPARE THE ROBOT
11  and STOP ALL MOTION
12  and MOVE BACKWARD
13  and HONK the horn for this many seconds .5
14  and WAIT this many seconds 1
15  and STOP ALL MOTION
16  and MOVE RANDOMLY
17  and WAIT RANDOMLY up to this many seconds 5
18  and STOP ALL MOTION
19  and PREPARE THE ROBOT
20  and CLEAR ALL ITEMS
21  and REPEAT THIS LOOP
22  and END   (Task is 8% full.)

X BETA (a Task)

 1 Does PREPARE THE ROBOT
 2  and INITIALIZE MEMORY
 3  and TURN ON THE FLASHING LIGHTS
 4  and BEGIN A LOOP
 5  and EXIT IF ANY BUMPER TOUCHED
 6  and REPEAT THIS LOOP
 7  and CALL this robot task line "BEGIN BETA HERE"
 8  and PREPARE THE ROBOT
 9  and CLEAR ALL ITEMS
10  and TURN ON THE FLASHING LIGHTS
11  and BEGIN A LOOP
12  and MOVE FORWARD
13  and EXIT IF ANY BUMPER TOUCHED
14  and REPEAT THIS LOOP
15  and MOVE WITH BETA INTELLIGENCE
16  and JUMP to the line called "BEGIN BETA HERE"
17  and END   (Task is 8% full.)
```

```
Z DO NUMBER TEST on <1>  (a Function)

 1 Does ADD the 0 and <1>
 2  and IF the SUM  IS  ERROR then
 3   Do    COPY from "FAIL" to RESULT
 4  and    COPY from <1> to ITEM NAME
 5  and OTHERWISE
 6   Do    COPY from "PASSED" to RESULT
 7  and    END of test
 8  and END   (Task is 5% full.)

Z EMPTY THE PROGRAM (a Task)

 1 Does ERASE all pages in folder SOURCE
 2  and ERASE all pages in folder LOOPS
 3  and ERASE all pages in folder VARIABLES
 4  and ERASE all pages in folder ERRORS
 5  and ERASE all pages in folder STRUCTURES
 6  and ERASE all pages in folder SPEECH LOAD DATA
 7  and Z INITIALIZE LEGAL VARIABLES
 8  and COPY from 0 to STATEMENT NUMBER
 9  and COPY from NO to SONAR NEEDED
10  and COPY from 0 to ERROR NUMBER
11  and COPY from NO to CHECKED
12  and COPY from NO to VOICE NEEDED
13  and COPY from NO to SONAR NEEDED
14  and COPY from EMPTY to RESULT
15  and COPY from EMPTY to SPEECH ADDRESS
16  and END   (Task is 8% full.)

Z ERROR CHECK IF/UNTIL STATEMENT (a Task)

 1 Does Z DO CHECK if item <1> is in folder "VARIABLES"
 2  and IF the RESULT  IS  "FAIL" then
 3   Do    Z SAVE FOLDER ERROR
 4  and OTHERWISE
 5   Do    GET the page indexed by <1> in folder VARIABLES
 6  and    COPY from VAR NAME to <1>
 7  and    END of test
 8  and Z DO CHECK if item <2> is in folder "CONDITIONS"
 9  and IF the RESULT  IS  "FAIL" then
10   Do    Z SAVE FOLDER ERROR
11  and OTHERWISE
12   Do    GET the page indexed by <2> in folder CONDITIONS
13  and    IF the TASK NAME  IS  "TEST IF" then
14   Do       COPY from OPPOSITE to <2>
15  and       END of test
16  and    END of test
17  and Z DO NUMBER TEST on <3>
18  and IF the RESULT  IS  "FAIL" then
19   Do    Z DO CHECK if item <3> is in folder "VARIABLES"
20  and    IF the RESULT  IS  "FAIL" then
21   Do       Z SAVE NUMBER OR FOLDER ERROR
22  and    OTHERWISE
23   Do       GET the page indexed by <3> in folder VARIABLES
24  and       COPY from VAR NAME to <3>
25  and       END of test
26  and    END of test
27  and END   (Task is 19% full.)
```

```
X BETA W/SONAR (a Task)

 1 Does PREPARE THE ROBOT
 2  and INITIALIZE MEMORY
 3  and TURN ON THE FLASHING LIGHTS
 4  and BEGIN A LOOP
 5  and EXIT IF ANY BUMPER TOUCHED
 6  and REPEAT THIS LOOP
 7  and CALL this robot task line "BEGIN BETA HERE"
 8  and PREPARE THE ROBOT
 9  and TURN ON THE FLASHING LIGHTS
10  and CLEAR ALL ITEMS
11  and BEGIN A LOOP
12  and MOVE FORWARD
13  and EXIT IF SONAR distance value is less than 95
14  and EXIT IF ANY BUMPER TOUCHED
15  and REPEAT THIS LOOP
16  and MOVE WITH BETA INTELLIGENCE
17  and JUMP to the line called "BEGIN BETA HERE"
18  and END   (Task is 9% full.)

X BUMPER ACTIVATED MOTIONS (a Task)

 1 Does DEFINE A VARIABLE called "A"
 2  and PREPARE THE ROBOT
 3  and WAIT this many seconds 5
 4  and BEGIN A LOOP
 5  and HONK the horn for this many seconds 2
 6  and WAIT this many seconds 1.5
 7  and LIGHTS ROUTINE
 8  and MOVE TIMED FORWARD for this many seconds 1.5
 9  and MOVE TIMED BACKWARD for this many seconds 1.5
10  and GO CLOCKWISE
11  and WAIT this many seconds 1.5
12  and GO COUNTERCLOCKWISE
13  and WAIT this many seconds 1.5
14  and STOP ALL MOTION
15  and HONK the horn for this many seconds 2
16  and WAIT this many seconds 1.5
17  and BEGIN A COUNTED LOOP called "A" beginning at 1 ending at 16
18  and SET the variable "@#7802" equal to "A"
19  and WAIT this many seconds .5
20  and REPEAT THIS COUNTED LOOP for the counter "A"
21  and STOP ALL MOTION
22  and SET the variable "@#7801" equal to 0
23  and BEGIN A LOOP
24  and EXIT IF ANY BUMPER TOUCHED
25  and REPEAT THIS LOOP
26  and REPEAT THIS LOOP
27  and END   (Task is 17% full.)
```

```
Z ERROR CHECK MATH FUNCTION (a Task)

 1 Does Z DO CHECK if item <1> is in folder "VARIABLES"
 2  and IF the RESULT  IS  "FAIL" then
 3   Do    Z SAVE FOLDER ERROR
 4  and OTHERWISE
 5   Do    GET the page indexed by <1> in folder VARIABLES
 6  and    COPY from VAR NAME to <1>
 7  and    END of test
 8  and Z DO NUMBER TEST on <2>
 9  and IF the RESULT  IS  "FAIL" then
10   Do    Z DO CHECK if item <2> is in folder "VARIABLES"
11  and    IF the RESULT  IS  "FAIL" then
12   Do       Z SAVE NUMBER OR FOLDER ERROR
13  and    OTHERWISE
14   Do       GET the page indexed by <2> in folder VARIABLES
15  and       COPY from VAR NAME to <2>
16  and       END of test
17  and    END of test
18  and Z DO CHECK if item <3> is in folder "FUNCTIONS"
19  and IF the RESULT  IS  "FAIL" then
20   Do    Z SAVE FOLDER ERROR
21  and    END of test
22  and Z DO NUMBER TEST on <4>
23  and IF the RESULT  IS  "FAIL" then
24   Do    Z DO CHECK if item <4> is in folder "VARIABLES"
25  and    IF the RESULT  IS  "FAIL" then
26   Do       Z SAVE NUMBER OR FOLDER ERROR
27  and    OTHERWISE
28   Do       GET the page indexed by <4> in folder VARIABLES
29  and       COPY from VAR NAME to <4>
30  and       END of test
31  and    END of test
32  and END   (Task is 22% full.)

Z ERROR CHECK STRUCTURES (a Task)

 1 Does BEGIN at page indexed by FIRST in folder STRUCTURES
 2  and START a loop
 3   Do LABEL this line "START A LOOP"
 4  and    GET the page indexed by NEXT in folder STRUCTURES
 5  and    IF the PAGE  IS  EMPTY then
 6   Do       LEAVE this loop
 7  and       END of test
 8  and    IF the CHECKED  IS  YES then
 9   Do       GO to the line called "START A LOOP"
10  and       END of test
11  and    IF the STRUCTURE TYPE  IS  "REPEAT" then
12   Do       Z SAVE STRUCTURE ERROR missing the statement "BEGIN" to go with t
e statment STRUCTURE TYPE at statement number STATEMENT NUMBER
13  and    OTHERWISE
14   Do       IF the STRUCTURE TYPE  IS  "END" then
15   Do          Z SAVE STRUCTURE ERROR missing the statement "IF OR ELSE" to g
 with the statment STRUCTURE TYPE at statement number STATEMENT NUMBER
16  and       OTHERWISE
17   Do          IF the STRUCTURE TYPE  IS  "NEXT" then
18   Do             Z SAVE STRUCTURE ERROR missing the statement "FOR" to go wi
h the statment STRUCTURE TYPE at statement number STATEMENT NUMBER
19  and          OTHERWISE
20   Do             IF the STRUCTURE TYPE  IS  "UNTIL" then
```

```
X CHARGER FINDER (a Task)

 1 Does WAIT this many seconds 20
 2  and PREPARE THE ROBOT
 3  and INITIALIZE MEMORY
 4  and BEGIN A LOOP
 5  and PREPARE THE ROBOT
 6  and CLEAR ALL ITEMS
 7  and BEGIN A LOOP
 8  and MOVE FORWARD
 9  and EXIT IF ANY BUMPER TOUCHED
10  and EXIT IF THE TAPE IS SENSED
11  and REPEAT THIS LOOP
12  and TEST IF the variable "R" is (=,<,>,<>) "=" compared to 0
13  and EXIT THIS LOOP
14  and END HERE
15  and MOVE WITH BETA INTELLIGENCE
16  and REPEAT THIS LOOP
17  and FOLLOW TAPE
18  and MAINTAIN CHARGE
19  and END   (Task is 7% full.)

X LOAD THE INTRODUCTION (a Task)

 1 Does REMARK that: "RB5X say the introduction - new voice cards only!"
 2  and LOAD THE PHRASE called: "SAY THE INTRODUCTION" with phonemes: "H.EH1.E
.L.O1.U1.AH1.EH3.I3.Y.PA1.PA1.AE1.EH3.M.THV.UH1.AH1.UH2.ER.B.E1.AY.Y.F.AH1.EH3
.V.PA1.EH1.EH2.K.PA0.S.I1.N.T.EH1.EH3.L.I1.I3.D.J.EH1.EH3.N.T.R.O1.U1.B.AH1.I
T.STOP."
 3  and END   (Task is 25% full.)

X SIMPLE SIMON (a Task)

 1 Does BEGIN A LOOP
 2  and PREPARE THE ROBOT
 3  and INITIALIZE VARIABLES
 4  and BEGIN A LOOP
 5  and EXIT IF FRONT+REAR BUMPER PRESS
 6  and REPEAT THIS LOOP
 7  and TURN ON THE FLASHING LIGHTS
 8  and HONK the horn for this many seconds .1
 9  and BEGIN A LOOP
10  and BEGIN A LOOP
11  and EXIT IF ANY BUMPER TOUCHED
12  and REPEAT THIS LOOP
13  and EXIT IF FRONT+REAR BUMPER PRESS
14  and WAIT this many seconds .2
15  and HONK the horn for this many seconds .1
16  and ASSIGN A MOTOR CODE
17  and TEST IF the variable "M" is (=,<,>,<>) ">" compared to "0"
18  and EXIT THIS LOOP
19  and END HERE
20  and CALCULATE variable "M" = "M" (+,-,*,/) "+"  1
21  and REPEAT THIS LOOP
22  and WAIT this many seconds 3
23  and BEGIN A COUNTED LOOP called "P" beginning at "N" ending at "M"
24  and SET the variable "@#7802" equal to "@P"
25  and WAIT this many seconds 1
26  and REPEAT THIS COUNTED LOOP for the counter "P"
27  and CLEAR ALL ITEMS
```

```
X SIMPLE SIMON (a Task)

28  and REPEAT THIS LOOP
29  and END    (Task is 16% full.)

Z COMPILE the BASIC statement <1> which means <2>  (a Function)

 1 Does Z INCREMENT STATEMENT NUMBER
 2  and COPY from <1> to BASIC TEXT
 3  and COPY from <2> to MEANING
 4  and SAVE new page in SOURCE
 5  and END    (Task is 2% full.)

Z COMPUTE 2 TO THE N-1 , with N = <1>  (a Function)

 1 Does COPY from 1 to PRODUCT
 2  and COPY from <1> to DIFFERENCE
 3  and START a loop
 4   Do     IF the DIFFERENCE  IS LESS THAN OR EQUAL TO  1 then
 5   Do        LEAVE this loop
 6  and     OTHERWISE
 7   Do        MULTIPLY the PRODUCT by 2
 8  and        SUBTRACT the 1 from DIFFERENCE
 9  and        END of test
10  and     REPEAT this loop
11  and END    (Task is 5% full.)

Z DELAY for this many loops:  <1>  (a Function)

 1 Does COPY from 0 to COUNTER
 2  and START a loop
 3   Do     Z INCREMENT the COUNTER
 4  and     IF the COUNTER  IS  <1> then
 5   Do        LEAVE this loop
 6  and        END of test
 7  and     REPEAT this loop
 8  and END    (Task is 3% full.)

Z DO CHECK if item <1> is in folder <2>  (a Function)

 1 Does ASSIGN the <2> to ASSIGNED FOLDER
 2  and WHEN folder ASSIGNED FOLDER  DOES NOT CONTAIN  <1> then
 3   Do     COPY from "FAIL" to RESULT
 4  and     COPY from <1> to ITEM NAME
 5  and     COPY from <2> to FOLDER NAME
 6  and OTHERWISE
 7   Do     COPY from "PASSED" to RESULT
 8  and     END of test
 9  and END    (Task is 5% full.)
```

```
Z ERROR CHECK STRUCTURES (a Task)

21   Do                    Z SAVE STRUCTURE ERROR missing the statement "DO" to r
ith the statment STRUCTURE TYPE at statement number STATEMENT NUMBER
22  and                 OTHERWISE
23   Do                    IF the STRUCTURE TYPE  IS  "EXIT" then
24   Do                       Z LOOK FOR BEGIN
25  and                    OTHERWISE
26   Do                       GET the page indexed by STRUCTURE TYPE in folder ST
MENT TYPES
27  and                      Z LOOK FOR MATCHING STATEMENT
28  and                      END of test
29  and                   END of test
30  and                END of test
31  and             END of test
32  and          END of test
33  and       REPEAT this loop
34  and END     (Task is 25% full.)

Z ERROR CHECK THE FOR STATEMENT (a Task)

 1 Does Z DO CHECK if item <1> is in folder "VARIABLES"
 2  and IF the RESULT  IS  "FAIL" then
 3   Do     Z SAVE FOLDER ERROR
 4  and OTHERWISE
 5   Do     GET the page indexed by <1> in folder VARIABLES
 6  and     COPY from VAR NAME to <1>
 7  and     END of test
 8  and Z DO NUMBER TEST on <2>
 9  and IF the RESULT  IS  "FAIL" then
10   Do     Z DO CHECK if item <2> is in folder "VARIABLES"
11  and     IF the RESULT  IS  "FAIL" then
12   Do        Z SAVE NUMBER OR FOLDER ERROR
13  and     OTHERWISE
14   Do        GET the page indexed by <2> in folder VARIABLES
15  and        COPY from VAR NAME to <2>
16  and        END of test
17  and     END of test
18  and Z DO NUMBER TEST on <3>
19  and IF the RESULT  IS  "FAIL" then
20   Do     Z DO CHECK if item <3> is in folder "VARIABLES"
21  and     IF the RESULT  IS  "FAIL" then
22   Do        Z SAVE NUMBER OR FOLDER ERROR
23  and     OTHERWISE
24   Do        GET the page indexed by <3> in folder VARIABLES
25  and        COPY from VAR NAME to <3>
26  and        END of test
27  and     END of test
28  and END     (Task is 19% full.)
```

```
Z ERROR CHECKING? (a Task)

 1 Does CARRIAGE return
 2  and DISPLAY the "Would you like your robot script checked for errors? (Y/N
"
 3  and LIMIT the input for ERROR CHECK to a maximum of 1
 4  and END   (Task is 7% full.)

Z FIND the label called <1>  (a Function)

 1 Does BEGIN at page indexed by FIRST in folder LOOPS
 2  and START a loop
 3   Do    GET the page indexed by NEXT in folder LOOPS
 4  and    IF the PAGE  IS  EMPTY then
 5   Do       IF the ERROR CHECK  IS  YES then
 6   Do          Z SAVE STRUCTURE ERROR missing the statement "CALL" to go wit
the statment "JUMP" at statement number TEMP STATEMENT NUMBER
 7  and          LEAVE this loop
 8  and          END of test
 9  and       LEAVE this loop
10  and       END of test
11  and    TAKE this many characters 1 from STRUCTURE TYPE
12  and    IF the FRONT  IS  "L" then
13   Do       IF the BACK  IS  <1> then.
14   Do          COPY from STATEMENT NUMBER to LABEL STATEMENT NUMBER
15  and          GET the page indexed by TEMP STATEMENT NUMBER in folder SOURCE
16  and          CUT the BASIC TEXT after "GOTO "
17  and          PASTE the FRONT in front of LABEL STATEMENT NUMBER
18  and          COPY from FRONT to BASIC TEXT
19  and          REPLACE this page in folder SOURCE
20  and          LEAVE this loop
21  and          END of test
22  and       END of test
23  and    REPEAT this loop
24  and END   (Task is 14% full.)

Z GET PREVIOUS BEGIN (a Task)

 1 Does GET the page indexed by THIS in folder LOOPS
 2  and IF the STRUCTURE TYPE  IS NOT  "BEGIN" then
 3   Do    START a loop
 4   Do       GET the page indexed by PREVIOUS in folder LOOPS
 5  and       IF the PAGE  IS  EMPTY then
 6   Do          LEAVE this loop
 7  and          END of test
 8  and       IF the STRUCTURE TYPE  IS  "BEGIN" then
 9   Do          LEAVE this loop
10  and          END of test
11  and       REPEAT this loop
12  and    END of test
13  and END   (Task is 6% full.)
```

```
Z GET PREVIOUS IF OR ELSE (a Task)

 1 Does COPY from EMPTY to STRUCTURE TYPE
 2  and GET the page indexed by THIS in folder LOOPS
 3  and IF the STRUCTURE TYPE  IS NOT  "IF" then
 4   Do    IF the STRUCTURE TYPE  IS NOT  "ELSE" then
 5   Do       START a loop
 6   Do          GET the page indexed by PREVIOUS in folder LOOPS
 7  and          IF the PAGE  IS  EMPTY then
 8   Do             LEAVE this loop
 9  and          OTHERWISE
10   Do             IF the STRUCTURE TYPE  IS  "IF" then
11   Do                LEAVE this loop
12  and             OTHERWISE
13   Do                IF the STRUCTURE TYPE  IS  "ELSE" then
14   Do                   LEAVE this loop
15  and                   END of test
16  and                END of test
17  and             END of test
18  and          REPEAT this loop
19  and       END of test
20  and    END of test
21  and END   (Task is 10% full.)

Z INCLUDE at <1> the statement <2> which means <3>  (a Function)

 1 Does COPY from <1> to STATEMENT NUMBER
 2  and COPY from <2> to BASIC TEXT
 3  and COPY from <3> to MEANING
 4  and SAVE new page in SOURCE
 5  and END   (Task is 2% full.)

Z INCREMENT the <1>  (a Function)

 1 Does ADD the 1 and <1>
 2  and COPY from SUM to <1>
 3  and END   (Task is 1% full.)

Z INCREMENT STATEMENT NUMBER (a Task)

 1 Does ADD the STATEMENT NUMBER and STATEMENT NUMBER INCREMENT VALUE
 2  and COPY from SUM to STATEMENT NUMBER
 3  and END   (Task is 1% full.)
```

```
Z INITIALIZE LEGAL VARIABLES (a Task)

 1 Does BEGIN at page indexed by FIRST in folder LEGAL VARIABLES
 2  and START a loop
 3   Do    GET the page indexed by NEXT in folder LEGAL VARIABLES
 4  and    IF the PAGE  IS  EMPTY then
 5   Do       LEAVE this loop
 6  and       END of test
 7  and    IF the USED  IS  NO then
 8   Do       LEAVE this loop
 9  and    OTHERWISE
10   Do       COPY from NO to USED
11  and       REPLACE this page in folder LEGAL VARIABLES
12  and       END of test
13  and    REPEAT this loop
14  and END   (Task is 6% full.)

Z INSURE ARM PULSE SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,400 then
 2   Do    COPY from STATEMENT NUMBER to HERE
 3  and    Z INCLUDE at 3,400 the statement "DELAY 1000:@#7807=#90" which mean
"SET STATUS BYTE"
 4  and    Z INCLUDE at 3,410 the statement "@#7806=#70" which means "ARM POWE
ON"
 5  and    Z INCLUDE at 3,420 the statement "@#7806=#F0" which means "SET ARM
R DRIVE"
 6  and    Z INCLUDE at 3,430 the statement "FOR I=1 TO P" which means "BEGIN
LOOP"
 7  and    Z INCLUDE at 3,440 the statement "@D=M:@D=N" which means "PULSE THE
RM"
 8  and    Z INCLUDE at 3,450 the statement "NEXT I" which means "REPEAT THIS
OP"
 9  and    Z INCLUDE at 3,460 the statement "@#7807=#90:@#7806=#B0" which means
"TURN OFF ARM POWER"
10  and    Z INCLUDE at 3,470 the statement "RETURN" which means "RETURN TO THE
MAIN PROGRAM"
11  and    COPY from HERE to STATEMENT NUMBER
12  and    END of test
13  and END   (Task is 36% full.)

Z INSURE BETA SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,160 then
 2   Do    COPY from STATEMENT NUMBER to ANSWER
 3  and    Z INCLUDE at 3,160 the statement "P=@#7800" which means "BUMPER PRES
ED"
 4  and    Z INCLUDE at 3,170 the statement "IF P=255 THEN P=251" which means "
ONAR TREATED AS BUMPER #1"
 5  and    Z INCLUDE at 3,180 the statement "V=@(N+P)" which means "CHECK EXPER
ENCE BLOCK"
 6  and    Z INCLUDE at 3,190 the statement "IF V<>#FF GOTO 3220" which means "
RY ACTION"
 7  and    Z INCLUDE at 3,200 the statement "V=RND(1,14)" which means "PICK RAN
OM ACTION"
 8  and    Z INCLUDE at 3,210 the statement "IF (V=3) OR (V=12) OR (V=(O+P)) GO
O 3200" which means "PICK ANOTHER"
 9  and    Z INCLUDE at 3,220 the statement "@#7802=V" which means "TRY ACTION"
10  and    WAIT this many seconds 2
11  and    IF the SONAR NEEDED  IS  YES then
```

```
Z INSURE BETA SUBROUTINE IS (a Task)

12   Do        Z COMPILE the BASIC statement "D=0" which means "DISTANCE EQUAL
 ZERO"
13  and        Z COMPILE the BASIC statement "LINK #1800" which means "CALL S
"
14  and        Z COMPILE the BASIC statement "@#7802=V" which means "CONTINUE
ION"
15  and     OTHERWISE
16   Do        Z COMPILE the BASIC statement "D=200" which means "CANCEL SONAR
ST"
17  and        END of test
18  and     Z COMPILE the BASIC statement "IF (@#7800<>#FF) OR (D<95) GOTO 336
which means "NOT SUCCESSFUL"
19  and     Z COMPILE the BASIC statement "@(N+P)=V" which means "SUCCESSFUL"
20  and     Z COMPILE the BASIC statement "CLEAR" which means "CLEAR ALL ITEMS
21  and     PASTE the "GOTO " in front of RETURN STATEMENT NUMBER
22  and     Z COMPILE the BASIC statement FRONT which means "BACK TO MAIN PRO
"
23  and     Z INCLUDE at 3,360 the statement "@(0+P)=V" which means "NOT SUCCE
UL"
24  and     Z INCLUDE at 3,370 the statement "GOTO 3200" which means ""
25  and     COPY from ANSWER to STATEMENT NUMBER
26  and     END of test
27  and END    (Task is 71% full.)

Z INSURE END OF PROGRAM IS THERE (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  2,999 then
 2   Do     END OF PROGRAM
 3  and     END of test
 4  and END    (Task is 1% full.)

Z INSURE PORT 7801 SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,100 then
 2   Do     COPY from STATEMENT NUMBER to HERE
 3  and     Z INCLUDE at 3,100 the statement "U=@#7801" which means "TURN ON A
T"
 4  and     Z INCLUDE at 3,105 the statement "U=U OR X" which means ""
 5  and     Z INCLUDE at 3,110 the statement "@#7801=U" which means ""
 6  and     Z INCLUDE at 3,115 the statement "RETURN" which means ""
 7  and     Z INCLUDE at 3,120 the statement "U=@#7801" which means "TURN OFF
IT @#7801"
 8  and     Z INCLUDE at 3,125 the statement "U=U AND X" which means ""
 9  and     Z INCLUDE at 3,130 the statement "@#7801=U" which means ""
10  and     Z INCLUDE at 3,135 the statement "RETURN" which means ""
11  and     COPY from HERE to STATEMENT NUMBER
12  and     END of test
13  and END    (Task is 24% full.)
```

```
Z INSURE RANDOM TURN SUBROUTINE (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,040 then
 2   Do    COPY from STATEMENT NUMBER to HERE
 3  and    Z INCLUDE at 3,040 the statement "V=RND(1,14)" which means "RANDOM
RN SUBROUTINE"
 4  and    Z INCLUDE at 3,045 the statement "IF (V=3) OR (V=12) GOTO 3040" whi
means ""
 5  and    Z INCLUDE at 3,050 the statement "RETURN" which means ""
 6  and    COPY from HERE to STATEMENT NUMBER
 7  and    END of test
 8  and END   (Task is 13% full.)

Z INSURE RANDOM WAIT SUBROUTINE (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,070 then
 2   Do    COPY from STATEMENT NUMBER to HERE
 3  and    Z INCLUDE at 3,070 the statement "T=RND(1,W)" which means "RANDOM WA
T SUBROUTINE"
 4  and    Z INCLUDE at 3,075 the statement "GOSUB 3000" which means ""
 5  and    Z INCLUDE at 3,080 the statement "RETURN" which means ""
 6  and    COPY from HERE to STATEMENT NUMBER
 7  and    Z INSURE WAIT SUBROUTINE IS
 8  and    END of test
 9  and END   (Task is 12% full.)

Z INSURE VOICE SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,500 then
 2   Do    COPY from STATEMENT NUMBER to HERE
 3  and    Z INCLUDE at 3,500 the statement "V=TOP+500+A:@V=P" which means "STO
E THE PHONEME"
 4  and    Z INCLUDE at 3,510 the statement "A=A+1" which means ""
 5  and    Z INCLUDE at 3,520 the statement "RETURN" which means ""
 6  and    COPY from HERE to STATEMENT NUMBER
 7  and    END of test
 8  and END   (Task is 11% full.)

Z INSURE WAIT SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE  DOES NOT CONTAIN  3,000 then
 2   Do    COPY from STATEMENT NUMBER to HERE
 3  and    Z INCLUDE at 3,000 the statement "FOR S=1 TO T" which means "WAIT T
ECONDS SUBROUTINE"
 4  and    Z INCLUDE at 3,005 the statement "DELAY 1000" which means ""
 5  and    Z INCLUDE at 3,010 the statement "NEXT S" which means ""
 6  and    Z INCLUDE at 3,015 the statement "RETURN" which means ""
 7  and    COPY from HERE to STATEMENT NUMBER
 8  and    END of test
 9  and END   (Task is 14% full.)
```

```
Z LOAD SONAR (a Task)

 1 Does DISPLAY the "Now loading the sonar subroutine..."
 2  and PRINT the ""
 3  and BEGIN at page indexed by FIRST in folder SONAR
 4  and START a loop
 5   Do    GET the page indexed by NEXT in folder SONAR
 6  and    IF the PAGE  IS  EMPTY then
 7   Do       LEAVE this loop
 8  and       END of test
 9  and    PASTE the SONAR ID in front of " "
10  and    PASTE the FRONT in front of BASIC TEXT
11  and    OUTPUT from FRONT using format DEFAULT
12  and    CARRIAGE return
13  and    DISPLAY the "$"
14  and    PRINT the ""
15  and    Z DELAY for this many loops:  5
16  and    REPEAT this loop
17  and Z DELAY for this many loops:  10
18  and PRINT the ""
19  and OUTPUT from "RUN" using format DEFAULT
20  and CARRIAGE return
21  and Z DELAY for this many loops:  50
22  and OUTPUT from "NEW#1841" using format DEFAULT
23  and CARRIAGE return
24  and Z DELAY for this many loops:  60
25  and OUTPUT from "NEW" using format DEFAULT
26  and CARRIAGE return
27  and DISPLAY the ""
28  and CARRIAGE return
29  and PRINT the ""
30  and END   (Task is 17% full.)

Z LOAD VOICE SUBROUTINE (a Task)

 1 Does CARRIAGE return
 2  and DISPLAY the "Now loading the voice subroutine..."
 3  and CARRIAGE return
 4  and PRINT the ""
 5  and BEGIN at page indexed by FIRST in folder VOICE SUBROUTINE
 6  and START a loop
 7   Do    GET the page indexed by NEXT in folder VOICE SUBROUTINE
 8  and    IF the PAGE  IS  EMPTY then
 9   Do       LEAVE this loop
10  and       END of test
11  and    OUTPUT from VOICE ID using format DEFAULT
12  and    CARRIAGE return
13  and    DISPLAY the "*"
14  and    PRINT the ""
15  and    Z DELAY for this many loops:  10
16  and    REPEAT this loop
17  and Z DELAY for this many loops:  10
18  and Z INSURE VOICE SUBROUTINE IS
19  and PRINT the ""
20  and OUTPUT from "@#2000=#03:@#2001=#0F:@#2002=#35:@#2003=#23:@#2004=#09" u
ng format DEFAULT
21  and CARRIAGE return
22  and Z DELAY for this many loops:  10
23  and OUTPUT from "@#2005=#21:@#2006=#1F:@#2007=#18:@#2008=#35:@#2009=#37:@"
0A=#1E" using format DEFAULT
```

```
Z LOAD VOICE SUBROUTINE (a Task)

24  and CARRIAGE return
25  and Z DELAY for this many loops:  10
26  and OUTPUT from "@#200B=#09:@#200C=#1E:@#200D=#3F:@#780B=#A7:S=#2000:LINK
F50" using format DEFAULT
27  and CARRIAGE return
28  and Z DELAY for this many loops:  10
29  and DISPLAY the ""
30  and CARRIAGE return
31  and PRINT the ""
32  and END   (Task is 33% full.)

Z LOOK FOR BEGIN (a Task)

 1 Does COPY from STATEMENT NUMBER to HERE
 2  and START a loop
 3   Do    GET the page indexed by PREVIOUS in folder STRUCTURES
 4  and    IF the PAGE  IS  EMPTY then
 5   Do       GET the page indexed by HERE in folder STRUCTURES
 6  and       Z SAVE STRUCTURE ERROR missing the statement "BEGIN" to go with
e statment "EXIT" at statement number HERE
 7  and       LEAVE this loop
 8  and       END of test
 9  and    IF the STRUCTURE TYPE  IS  "BEGIN" then
10   Do       GET the page indexed by HERE in folder STRUCTURES
11  and       COPY from YES to CHECKED
12  and       REPLACE this page in folder STRUCTURES
13  and       LEAVE this loop
14  and       END of test
15  and    REPEAT this loop
16  and END   (Task is 9% full.)

Z LOOK FOR MATCHING STATEMENT (a Task)

 1 Does COPY from STATEMENT NUMBER to HERE
 2  and START a loop
 3   Do    GET the page indexed by NEXT in folder STRUCTURES
 4  and    IF the PAGE  IS  EMPTY then
 5   Do       GET the page indexed by HERE in folder STRUCTURES
 6  and       Z SAVE STRUCTURE ERROR missing the statement MATCHING STATEMENT t
go with the statment STATEMENT TYPES ID at statement number STATEMENT NUMBER
 7  and       LEAVE this loop
 8  and       END of test
 9  and    IF the CHECKED  IS NOT  YES then
10   Do       IF the STRUCTURE TYPE  IS  MATCHING STATEMENT then
11   Do          COPY from YES to CHECKED
12  and          REPLACE this page in folder STRUCTURES
13  and          GET the page indexed by HERE in folder STRUCTURES
14  and          COPY from YES to CHECKED
15  and          REPLACE this page in folder STRUCTURES
16  and          LEAVE this loop
17  and          END of test
18  and       END of test
19  and    IF the STATEMENT TYPES ID  IS  "IF" then
20   Do       IF the STRUCTURE TYPE  IS  "ELSE" then
21   Do          GET the page indexed by HERE in folder STRUCTURES
22  and          Z SAVE STRUCTURE ERROR missing the statement MATCHING STATEMEN
 to go with the statment STATEMENT TYPES ID at statement number STATEMENT NUMBE
```

```
Z LOOK FOR MATCHING STATEMENT (a Task)

23  and           LEAVE this loop
24  and           END of test
25  and        END of test
26  and     REPEAT this loop
27  and END    (Task is 16% full.)

Z OUTPUT PAGE NUMBER (a Task)

 1 Does PASTE the "page " in front of PAGE COUNT
 2  and OUTPUT from SPACES using format 36
 3  and OUTPUT from FRONT using format DEFAULT
 4  and CARRIAGE return
 5  and CARRIAGE return
 6  and END    (Task is 2% full.)

Z OUTPUT PROGRAM HEADING (a Task)

 1 Does CARRIAGE return
 2  and OUTPUT from "     Stmt #" using format 10
 3  and OUTPUT from "   Tiny BASIC Text          " using format 30
 4  and OUTPUT from "        Meaning of Text" using format 23
 5  and CARRIAGE return
 6  and CARRIAGE return
 7  and END    (Task is 8% full.)

Z PATCH UP END STATEMENTS (a Task)

 1 Does COPY from STATEMENT NUMBER to END STATEMENT NUMBER
 2  and Z GET PREVIOUS IF OR ELSE
 3  and DELETE the page indexed by STATEMENT NUMBER from folder LOOPS
 4  and IF the STRUCTURE TYPE  IS  "IF" then
 5   Do    GET the page indexed by STATEMENT NUMBER in folder SOURCE
 6  and    CUT the BASIC TEXT after "GOTO "
 7  and    PASTE the FRONT in front of EXIT STATEMENT NUMBER
 8  and    COPY from FRONT to BASIC TEXT
 9  and    REPLACE this page in folder SOURCE
10  and OTHERWISE
11   Do    IF the STRUCTURE TYPE  IS  "ELSE" then
12   Do       SUBTRACT the 25 from STATEMENT NUMBER
13  and       COPY from DIFFERENCE to STATEMENT NUMBER
14  and       PASTE the "GOTO " in front of END STATEMENT NUMBER
15  and       Z COMPILE the BASIC statement FRONT which means "SKIP THE ELSE"
16  and       END of test
17  and    END of test
18  and COPY from END STATEMENT NUMBER to STATEMENT NUMBER
19  and END    (Task is 13% full.)
```

```
Z PATCH UP EXIT STATEMENTS (a Task)

 1 Does COPY from STATEMENT NUMBER to EXIT STATEMENT NUMBER
 2  and START a loop
 3   Do    GET the page indexed by NEXT in folder LOOPS
 4  and    IF the PAGE  IS  EMPTY then
 5   Do       LEAVE this loop
 6  and    OTHERWISE
 7   Do       IF the STRUCTURE TYPE  IS  "EXIT" then
 8   Do          GET the page indexed by STATEMENT NUMBER in folder SOURCE
 9  and          CUT the BASIC TEXT after "GOTO "
10  and          PASTE the FRONT in front of EXIT STATEMENT NUMBER
11  and          COPY from FRONT to BASIC TEXT
12  and          REPLACE this page in folder SOURCE
13  and          DELETE the page indexed by STATEMENT NUMBER from folder LOOPS
14  and       OTHERWISE
15   Do          LEAVE this loop
16  and          END of test
17  and       END of test
18  and    REPEAT this loop
19  and COPY from EXIT STATEMENT NUMBER to STATEMENT NUMBER
20  and END   (Task is 10% full.)

Z PATCH UP JUMP STATEMENTS (a Task)

 1 Does BEGIN at page indexed by FIRST in folder LOOPS
 2  and START a loop
 3   Do    GET the page indexed by NEXT in folder LOOPS
 4  and    IF the PAGE  IS  EMPTY then
 5   Do       LEAVE this loop
 6  and       END of test
 7  and    TAKE this many characters 1 from STRUCTURE TYPE
 8  and    IF the FRONT  IS  "J" then
 9   Do       COPY from STATEMENT NUMBER to TEMP STATEMENT NUMBER
10  and       COPY from BACK to LABEL NAME
11  and       Z FIND the label called LABEL NAME
12  and       GET the page indexed by TEMP STATEMENT NUMBER in folder LOOPS
13  and       DELETE the page indexed by THIS from folder LOOPS
14  and       END of test
15  and    REPEAT this loop
16  and END   (Task is 8% full.)

Z SAVE FOLDER ERROR (a Task)

 1 Does IF the ERROR CHECK  IS  YES then
 2   Do    ADD the STATEMENT NUMBER and 10
 3  and    COPY from SUM to LINE NUMBER
 4  and    PASTE the "Folder " in front of FOLDER NAME
 5  and    PASTE the FRONT in front of " doesn't contain the item "
 6  and    PASTE the FRONT in front of ITEM NAME
 7  and    COPY from FRONT to ERROR MESSAGE
 8  and    Z INCREMENT the ERROR NUMBER
 9  and    SAVE new page in ERRORS
10  and    END of test
11  and END   (Task is 9% full.)
```

```
Z SAVE NUMBER OR FOLDER ERROR (a Task)

 1 Does IF the ERROR CHECK  IS  YES then
 2   Do    ADD the STATEMENT NUMBER and 10
 3  and    COPY from SUM to LINE NUMBER
 4  and    PASTE the ITEM NAME in front of " should be a number or be containe
in the folder "
 5  and    PASTE the FRONT in front of FOLDER NAME
 6  and    COPY from FRONT to ERROR MESSAGE
 7  and    Z INCREMENT the ERROR NUMBER
 8  and    SAVE new page in ERRORS
 9  and    END of test
10  and END   (Task is 10% full.)

Z SAVE STRUCTURE ERROR missing the statement <1> to go with the statment <2>
statement number <3>  (a Function)

 1 Does PASTE the "The " in front of <2>
 2  and PASTE the FRONT in front of " statement is missing the "
 3  and PASTE the FRONT in front of <1>
 4  and PASTE the FRONT in front of " statement"
 5  and COPY from FRONT to ERROR MESSAGE
 6  and COPY from <3> to LINE NUMBER
 7  and Z INCREMENT the ERROR NUMBER
 8  and SAVE new page in ERRORS
 9  and IF the <1>  IS NOT  "CALL" then
10   Do    COPY from YES to CHECKED
11  and    REPLACE this page in folder STRUCTURES
12  and    END of test
13  and END   (Task is 11% full.)

ZZ AVAIL TASK (a Task)

 1 Does END   (Task is 1% full.)

ZZ AVAIL TASK #1 (a Task)

 1 Does END   (Task is 1% full.)

ZZ AVAIL TASK #10 (a Task)

 1 Does END   (Task is 1% full.)
```

```
ZZ AVAIL TASK #11 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #12 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #13 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #14 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #2 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #3 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #4 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #5 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #6 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #7 (a Task)
 1 Does END    (Task is 1% full.)
ZZ AVAIL TASK #8 (a Task)
 1 Does END    (Task is 1% full.)
```

ROBOT CONTROL LANGUAGE(tm) WITH SAVVY(R)

TUTORIAL

GENERAL ROBOTICS CORPORATION Golden, Colorado

1986

ROBOT CONTROL LANGUAGE WITH SAVVY
TUTORIAL

TABLE OF CONTENTS

A. INTRODUCTION

Congratulations on your purchase of Robot Control Language(tm) with Savvy(R) - RCL(tm), for short. Developed jointly by Robot Corporation and Excalibur Technologies Corporation, RCL is more than just another programming language. RCL is a complete software development system that virtually eliminates the language barrier between robots and people by allowing you to program your robot using common English words and phrases.

This Tutorial introduces you to the fun of programming with RCL and explains some of the terms you need to know as you begin creating your own RCL programs. As with other Tutorials, we suggest you use this one while seated at your computer, with your RB5X nearby, so that you can do the examples as we go along. For details about RCL not covered in this Tutorial, see the RCL Programmer's Manual that comes with this package.

WHAT YOU GET WHEN YOU ORDER RCL WITH SAVVY

To start, let's look at what you get when you order the RCL package.

In addition to this Tutorial, you receive a Robot Control Language with Savvy Programmer's Manual and the Savvy Personal Language Reference Folio. (We will discuss the relationship of RCL to Savvy a little later in this Tutorial.)

We also provide the Savvy circuit card, which you need to use RCL and which fits into Slot 7 inside your Apple computer.

If you ordered RCL I, you received one RCL diskette, which contains RCL with Savvy and some initial robot tasks we've developed; a Savvy One Master diskette, which enables you to use Savvy alone for other purposes on your Apple; and a "Let's Use Savvy" diskette. With this system, you store the robot tasks you create on the RCL diskette; it is a good idea to make several copies of the diskette before you begin designing robot tasks so you will have additional diskette space for storing the robot tasks you develop later.

If you ordered RCL II, you received one RCL diskette, which contains RCL and the initial robot tasks; one Savvy Pro Master diskette, which you will use with the RCL diskette (or with a blank data disk, if you wish to use Savvy with your Apple for other purposes); a Savvy "Demo" diskette; and a "Let's Use Savvy" diskette.

The RCL I diskette or the Savvy Pro and RCL II diskettes should always be in the disk drives when RCL is in use. They are standard floppy diskettes; the Savvy manual tells you how to handle and care for them.

RCL I has everything for programming robot tasks that RCL II has except the Savvy "Suspend" and selective file transfer capabilities, which were excluded to provide more space in the one-diskette system. For more information on these capabilities, consult your Savvy Personal Language Reference Folio.

WHAT YOU NEED TO GET STARTED

RCL I

To set up RCL I, you need the following hardware items:

Most important, of course, is one RB5X robot, which comes with its own Reference Manual.

You also need an Apple II+ or Apple IIe computer with a single disk drive and controller.

To establish communication between your computer and the RB5X, you must have a serial communications card -- either Apple's Super Serial Card or California Computer Systems (CCS) 7710A.

You also need an RS-232 cable for loading the RCL programs you write using your Apple into the RB5X. The connector on one end should be a 25-pin D male connector. (If this sounds like Greek to you, ask your computer dealer for help in assuring that you have the right cable.) This goes into Port 1 on the back of your RB5X. The other end must match the RS-232 port on your Apple serial card.

Insert the CCS 7710A or the Super Serial card in Slot 1 in your Apple and set the card's baud rate at 1200. Slot 1 is commonly used for a printer; when using RCL, your computer communicates with RB5X as though it were a printer. Therefore, when loading robot tasks into RB5X you cannot also have a printer connected to your Apple.

When using the CCS 7710A card you must also make the following modifications to establish proper serial communication:

1. Switch RS-232 lines 2 and 3 (inside the robot). This is done by positioning the switch on the RB5X's interface panel circuit board up, instead of down (normal).

2. Connect pin 4 on the CCS card to pin 6 on the CCS card.

3. Make sure there is no connection between pin 25 on the CCS card and the robot. Severe damage to the card and RB5X may result from this connection.

Please see your RB5X dealer if you need help making these cable modifications.

You may want to use an 80-column display card because it allows you to use both upper and lower case letters, making the screen easier to read. If you have an Apple IIe, any brand of card will do. If you have an Apple II+, you must use the Videx Videoterm card. This card should be inserted in Slot 3 of your Apple. Refer to pages 3-4 of your Savvy manual for further information on hardware set up.

RCL II

RCL II has the same hardware requirements as RCL I, with one difference. To use this system, you need from two to four disk drives or, optionally, a hard disk drive.

RB5X OPTIONS

You might want to add options -- such as the RB arm or voice/sound synthesis -- to your RB5X to make your programming in RCL more fun and more challenging. The sample program we use in this Tutorial, for example, works with an RB5X that has voice capability.

WHAT EXACTLY IS RCL WITH SAVVY?

RB5X's native tongue is Tiny BASIC. But as we all know, computer languages such as Tiny BASIC can be difficult for people to learn. There are complicated codes and symbols to memorize, and every character you enter must be in its proper place. No extra spaces, no transposed letters. Otherwise, it isn't Tiny BASIC and RB5X can't understand what you want it to do. And while Tiny BASIC is easier to learn than many programming languages, it can still be frustrating.

So RCL's primary responsibility is to act as a foreign language interpreter, translating what you say in your own words into the Tiny BASIC in which your RB5X is fluent.

RCL is based on Excalibur's personal language system called Savvy. So to understand RCL, you must have at least some understanding of Savvy.

Think of it this way: RCL is a subset of Savvy; it exists within Savvy, as your house exists within its neighborhood. RCL has its own personality, its own unique style of architecture. But it also draws some of its characteristics from its environment, from its neighborhood; in this case, from Savvy.

So what is Savvy? It really is a lot of things. It is an operating system -- you don't need CP/M or Apple DOS or any other operating system.

Savvy is a Virtual Resources Manager that lets your small microcomputer run huge programs as if it were a much larger machine.

Savvy is a language. More specifically, Savvy learns your language from you -- whether it is English, French, or Turkish. But since it needs some language to start with, Savvy learns an important primary vocabulary at the factory.

Savvy is an associative memory, which lets the system place items in its memory based on their description; therefore, the system will find what you want if you describe it. You don't have to assign or ask for an actual memory address, which is a complicated programming activity.

Savvy is an automatic program generator, which allows it to write "families" of programs for you, designing them for specific purposes.

But, most important, Savvy is a personality. You communicate with it using words instead of baffling, hard-to-remember codes.

Using Savvy as its basis, RCL then learns your language from you and offers RB5X owners a flexible, interactive, conversational method of writing programs for the robot.

In addition to learning your language from you, RCL also tries to figure out what you mean by what you enter at the keyboard. For example, if you enter the words MVOE FOREWARD on your computer keyboard exactly as we have written here, errors and all, RCL searches its memory trying to figure out what you really intended to enter. It then presents you with as many phrases as it has that resemble MVOE FOREWARD and asks you which one you want:

I'm not sure what MVOE FOREWARD refers to.
Is it one of these:
1. MOVE FORWARD
2. MOVE BACKWARD
3. MOVE RANDOMLY
4. or None of the above?

Please type the number of the correct answer:

RCL also comes with some standard tasks that you can incorporate into the programs you write yourself for your RB5X. (MOVE FORWARD is one of those tasks.)

So RCL is an intelligent software development system -- able to draw conclusions, willing to overlook your spelling errors, and ready to learn your language from you -- that is just perfect for use with the RB5X Intelligent Robot!

SOME USEFUL DEFINITIONS

Before we actually step you through a sample program, there are some important definitions that will help you to use RCL.

Robot Tasks and Primary Tasks

In RCL, a "robot task" is a program that you write in order to train RB5X to do something -- for example, move through the living room using its sonar and tactile sensors to detect objects in its way.

A robot task is composed of smaller, primary tasks, some of which we have already written for you to use in the composition of your robot tasks and some of which you will create yourself. So unlike other languages where you have routines and subroutines as distinct from procedures and functions, in RCL, everything is a task. A robot task simply comprises a group of shorter tasks, one nested neatly inside the other.

A list of the provided RCL tasks appears at the end of this Tutorial, in Appendix 3. Take a look now at this list. You will notice that we have starred (*) some of these tasks. This designates the tasks with which everyone using RCL should be familiar, whether they are writing simple or complex robot tasks. The other, unstarred entries are tasks that will be used by more experienced users of RCL.

We use some of the starred entries in the "Sample Robot Task" later in this Tutorial.

Prompt

There are times when you are working in RCL that the system prompts you. This is a common programming term meaning that a message appears on the screen asking you for more information or instructing you on what to do next.

The RCL prompt you probably see most often is:

What would you like me to do now?

as the system awaits your next set of instructions.

Menus

While you are working in RCL, you will notice that the system periodically displays lists of options, like a restaurant menu, from which you choose items for the system to accomplish. For example, if you use the Savvy SUMMON command, the system displays a menu and gives you several choices: you can either backup your diskettes, configure the system for your individual computer setup, or run diagnostics. You also have the choice of returning to the "What would you like me to do now?" prompt.

Section B, "Getting Started," contains several examples of RCL menus.

Replacement Symbols

You will notice as you look through the list of primary tasks that some of them contain angle brackets (< >) enclosing numbers. These numbers designate replacement symbols, characters that stand for information that you must supply.

For example, in the task "DEFINE A VARIABLE called <1>", the number 1 is the replacement symbol. You must supply the name of the variable -- for example, DEFINE A VARIABLE called "A". Note that when you supply the answer in place of the replacement symbol, unless the answer is a number (value) or unless it is a response to the direction prompt in the commands for the RB arm (OPEN, OUT, etc.), it must be enclosed in quotation marks. You type the first set of quotations (") and when you press <RETURN>, Savvy supplies the ending quotation marks.

Loops

Another concept you must understand is loops. If you have done any programming, you should recognize this term. Even if you haven't, you may have heard it and have a vague idea of what it means.

In its most fundamental sense, a loop is a procedure that a computer program -- or, in this case, a robot task -- repeats over and over again until it meets some condition or set of circumstances, allowing the program to stop the repetition. In other words, a loop within a robot task instructs your RB5X to repeat a procedure over and over again until some condition is met that allows it to stop. And if you don't designate this condition, your RB5X repeats and repeats until you switch it off. Our "Sample Robot Task" later in this Tutorial includes several loops and illustrates this concept.

In RCL, there are three kinds of loops: limited loops, counted loops, and general loops.

The limited loop is identified in RCL as the "BEGIN A LIMITED LOOP" task, and is ended with the command "REPEAT THE LIMITED LOOP until <1> is (<,>,=,<>) <2> to <3>." In a limited loop (which Tiny BASIC programmers will recognize as a DO/UNTIL loop), the robot task repeats the same procedure over and over again until it encounters a condition that allows it to leave the loop. This condition is defined by the "REPEAT THE LIMITED LOOP" command.

In a counted loop (which Tiny BASIC programmers will recognize as a FOR/NEXT loop), the robot task repeats a procedure for a specified number of times. The task "BEGIN A COUNTED LOOP" instructs the robot task to start this procedure and begin counting the times it is repeated. For this task, your computer system prompts you for the name of the loop, where it begins, and where it ends. So the task reads "BEGIN A COUNTED LOOP called <1> beginning at <2> ending at <3>". Remember that 1, 2, and 3 are replacement symbols for information you must supply. The number 1 represents a variable name that will be used in Tiny BASIC, so it must be a single letter enclosed in quotation marks. The numbers 2 and 3 designate variable values that RCL counts from the first value to the second, one number at a time. For example, the loop that begins with "BEGIN A COUNTED LOOP called "p" beginning at 1 and ending at 5" and is closed with "REPEAT THIS COUNTED LOOP for the counter "p"" will be executed five times.

General (unconditional) loops (which Tiny BASIC programmers will recognize as a GOTO construction) will repeat from "BEGIN A LOOP" to "REPEAT THIS LOOP" until some condition within the loop causes the task to exit and proceed with the instructions that follow. The primary task "TEST IF the variable <1> is (=,<,>,<>) <2> compared to <3>" and the primary tasks that begin with EXIT are most often used to set up a condition within the loop.

SAVVY DEFINITIONS

Since RCL is a subset of Savvy, you occasionally need to use some commands that are basic Savvy functions and are not really additional features of RCL. We have listed some of these commands -- called primaries -- here, but you should examine your Savvy manual for details on using the Savvy primaries.

ASSOCIATE

This command enables you to teach RCL your language. Specifically, you can tell it to "associate" one phrase with another. For example, using this task, you can teach RCL to think of the two tasks "CLEAN THE CARPET" and "VACUUM THE FLOOR" as meaning exactly the same thing.

DEFINE

This is the command you use to create robot tasks in RCL. The "Sample Robot Task" beginning on page 21 begins with DEFINE.

DETAIL

This command allows the user to list the various components of a robot task. For example, if you look at the "BEGIN A LOOP" detail in Appendix 2, you see the four separate tasks that comprise "BEGIN A LOOP."

EDIT

This command allows you to change a robot task you already created using the DEFINE command. In using this command, you have access to several important functions, including L (which allows you to list the task you are editing), <#>--a line number (which enables you to automatically move the cursor to the line number specified), D (which deletes the line above your cursor), and Q (which allows you to leave, or quit the EDIT task and return to the mode where you can enter more robot tasks). The "Sample Robot Task" in Section C describes how to use these functions.

FORGET

This command cancels an association that you may have previously created using the ASSOCIATE command. For example, if you told the system to recognize "CLEAN THE CARPET" and "VACUUM THE FLOOR" as meaning the same thing, and later decide they should have two separate meanings, you would use this command.

RENAME

This command allows the user to rename a task throughout the RCL system. This is most often used when customizing RCL to your particular needs.

SUMMON

The SUMMON command allows you to display on the screen the EXPLANATION of STANDBY TASKS menu (RCL II) or the UTILITIES menu (RCL I). From these menus, you can choose to make backups of your diskettes, configure RCL to match your computer system, or run a diagnostic check of your system.

SOME KEYBOARD PRELIMINARIES

In general, your Apple keyboard is similar to the keyboard of a standard typewriter, with a few extra keys added that correspond to specific computer functions.

Your Apple User's Manual contains detailed information about the operation of the computer itself, including the functions of the various keys. But it is important that you understand how RCL handles certain keys.

RESET Key

The RESET key interrupts the computer's CPU and leaves no way for you to return to what you were doing. As a rule, you should **NEVER** use the RESET key as it will cause irreparable damage to your RCL diskette or diskettes.

RETURN Key

Any time you type an instruction in RCL, you need to end it by pressing the RETURN key. This signals the system to look at what you have entered. In the sample program in this Tutorial, we use <RET> to remind you to press the RETURN key.

Left-Pointing Arrow

If you make a mistake while entering RCL instructions, use the left-pointing arrow to back up the cursor to the mistake. Then type your correction. Each time you press this key, the cursor backs up one character, erasing that character.

Right-Pointing Arrow

The right-pointing arrow acts as a "delete" key. When you press this key, you delete everything you have typed since you last pressed the RETURN key.

Escape Key

Use the Escape key, labeled ESC on the left side of your keyboard, to stop an operation. Pressing this key returns you to the "Finish, Resume, or Suspend?" prompt. Press the first letter of the choice you wish to make to continue from there. See your Savvy Personal Language Reference Folio for a detailed description of the "Finish, Resume, or Suspend?" function.

Upper-Case Letters

You should remember to type all of your RCL instructions in capital or upper-case letters. The system then responds or prompts you in all lower-case letters. (That is, unless you are working with an Apple II+ with a 40-column display. In this case, everything on the screen appears in upper-case letters.) The shift key does not work with Savvy; to shift between upper and lower case input press CTL and A at the same time.

B. GETTING STARTED

One of the first things you should do when you open your RCL package is to configure the diskettes to your individual computer system and then make copies of the diskette or diskettes.

In RCL I, the one-diskette system, the system software takes up about 50% of the diskette space, leaving you only 50% for your own program development. So it is a good idea to make several copies of your RCL I diskette and to use these copies for writing different RCL programs for your robot.

With RCL II, the Savvy system software is contained on one diskette, leaving the second diskette for RCL commands and your program development. However, it is still a good idea to make copies of both the system diskette and the RCL diskette so that if your system ever malfunctions while your diskettes are in the drives, you can recreate the work you did up until the system failure.

TO CONFIGURE RCL FOR YOUR SYSTEM

RCL I

To configure RCL I for your system:

1. Make sure the Savvy circuit card is in Slot 7 inside your Apple and that the serial card is in Slot 1.

2. Insert your RCL diskette, with its label up and the notch to the left, into the Apple disk drive and close the door.

3. Switch the Apple on.

4. At the "What would you like me to do now?" prompt, use the Savvy SUMMON command to display the UTILITIES menu. Type:

   SUMMON<RET>

   The system then prompts you with the message:

   the standby task

5. Type:

"<RET>

and the system displays the following menu:

--UTILITIES MENU--

1. Make an exact copy of this diskette.

2. Perform system DIAGNOSTICS.

3. CONFIGURE for different hardware.

4. ALL DONE WITH HELP...Go back to 'What would you like me to do now?'

ENTER the NUMBER of the desired action:

6. Choose the CONFIGURE option by pressing:

3

There is no need to press the RETURN key.

The system then displays the following screen:

WHICH CRT (MONITOR) WILL YOU BE USING?
(ASTERISK SHOWS PRESENT CONFIGURATION)

*1) 40-COLUMN NATIVE APPLE SCREEN WITHOUT LOWER-CASE.

2) 40-COLUMN NATIVE APPLE SCREEN WITH LOWER CASE. (SPECIAL EQUIP. OR II-E REQ'D)

3) 80-COLUMN VIDEX VIDEOTERM. (SPECIAL EQUIP. REQ'D)

4) 80-COLUMN NATIVE APPLE SCREEN. (II-E REQ'D)

ENTER THE NUMBER FOR DESIRED CRT:

7. Press the number key that corresponds to your configuration. The system then displays the following menu:


```
WHICH PRINTER INTERFACE WOULD YOU LIKE?
(ASTERISK SHOWS PRESENT CONFIGURATION)

 1) APPLE PARALLEL

*2) APPLE SUPER SERIAL

 3) CCS 7728 PARALLEL

 4) CCS 7710 SERIAL

 5) NO PRINTER PRESENT

ENTER THE NUMBER FOR DESIRED PRINTER.
```

8. Remember that you won't actually be using a printer. RCL simply thinks of the RB5X as a printer. Press the number key that corresponds to your printer setup (either #2 or #4). The system then returns you to the "What would you like me to do now?" prompt.

Once you have configured your diskette to match your system, you are ready to move on to the section, "Backing Up Your Diskettes."

RCL II

To configure RCL II for your system:

1. Make sure the Savvy circuit card is in Slot 7 inside your Apple and that the serial card is in Slot 1.

2. Insert the Savvy system diskette with its label up and the notch to the left into Disk Drive 1, and close the door.

3. Insert your RCL II diskette with its label up and the notch to the left into Disk Drive 2, and close the door.

4. Switch the Apple on.

5. At the "What would you like me to do now?" prompt, use the Savvy SUMMON command to display the EXPLANATION of STANDBY TASKS menu. Type:

   ```
   SUMMON<RET>
   ```

   The system prompts you with the message:

   ```
   the standby task
   ```

6. Type:

   ```
   "<RET>
   ```


and the system displays the following menu:

EXPLANATION of STANDBY TASKS

1. ARCHIVE is used to make copies of disks. It can also be used to do special disk related operations.

2. CONFIGURATOR is used to tell SAVVY what hardware is in your system and to make clean data disks.

3. DIAGNOSTICS is used to test the integrity of the data on your data disk. It can also clear the 'your disk may be in trouble' message.

4. All done with UTILITIES...Go back to 'What would you like me to do now?'

ENTER the NUMBER of desired action:

7. Choose the CONFIGURATOR option by pressing:

2

There is no need to press the RETURN key.

8. The system then displays the following menu:

-- CONFIGURATOR MENU --

1) Make a CLEAN new DATA-DISK.

2) Change the Hardware Configuration. (Monitors, Printers, Disk-Drives)

3) Extend the length of the DATA-DISK.

4) Shorten the length of the DATA-DISK.

5) All done with the CONFIGURATOR...go back to 'What would you like to do now?'

ENTER the NUMBER for the desired action:

Choose the second option by pressing:

2

There is no need to press the RETURN key.
The system then displays the following message:

You may VIEW the CURRENT configuration
of the hardware by responding with only
a carriage RETURN to the questions.

HOWEVER, if you make any configuration
changes, I will automatically RESTART
to make your change become effective.

Press any key to continue.

9. Press any key to continue. The system then displays the following menu:

Which CRT (Monitor) will you be using?
(Asterisk shows present configuration.)

1) 40 column native APPLE screen
without lower-case

*2) 40 column native APPLE screen
with lower-case
(Special equip. or II-e req'd.)

3) 80 column VIDEX videoterm
(Special equip. req'd.)

4) 80 column native APPLE screen.
(II-e req'd.)

ENTER the NUMBER for desired CRT:

10. Press the number key that corresponds to your system and the following menu appears:

Which printer interface would you like?
(Asterisk shows present configuration.)

1) APPLE parallel

*2) APPLE Super Serial

3) CCS 7728 parallel

4) CCS 7710 serial

5) No printer present

ENTER the NUMBER for desired PRINTER:

11. Remember that you won't actually be using a printer. RCL simply thinks of the RB5X as a printer. Press the number key that corresponds to your system (either #2 or #4) and the following menu appears:

What drive will you be using for DATA?
(Asterisk shows present configuration)

*1) Floppy diskettes

2) Corvus hard-disk

The MASTER-DISK you are now using is presently configured for a data-disk on Drive #2.

ENTER NUMBER for desired data-drive:

12. Press the number key that corresponds to your system and the following message appears:

    I always use diskette Drive #1 for the SAVVY MASTER diskette.

    How many of the REMAINING Drives shall I use for DATA storage (1-3)?

13. Press the number key that corresponds to your system and the following message appears to confirm your choice:

    For DATA I'll use Drive #X.

    Press any key to continue.

    where the X stands for the number 1 through 3 that you choose.

14. Press any key to continue and the following message appears, signaling you that configuration is complete.

    I will now RESTART to make these changes become effective immediately.

    If you changed your CRT configuration, the following message also appears:

    Reminder...you changed your video.
    You may need to move a cable.

The system then returns to the "What would you like me to do now?" prompt, and you may now move on to the section, "Backing Up Your Diskettes."

Once you finished configuring your diskette to match your computer system, your screen displays the "What would you like me to do now?" prompt. To make backups of your diskettes:

1. Use the Savvy SUMMON command to display the UTILITIES menu again. Enter:

   SUMMON<RET>

   The system then prompts you with the message:

   the standby task

2. Type:

   "<RET>

   The system displays the UTILITIES menu:

   --UTILITIES MENU--

   1. Make an exact copy of this diskette.

   2. Perform system DIAGNOSTICS.

   3. CONFIGURE for different hardware.

   4. ALL DONE WITH HELP...Go back to 'What would you like me to do now?'

   ENTER the NUMBER of the desired action:

3. To choose the backup (or copy) option from this menu, press:

   1

   The system then displays the following message:

   This procedure allows you to make an exact copy of this ORIGINAL. You will need a spare diskette to continue.

   Do you still want to copy?

4. Type:

   Y

for yes and the system displays the following prompt:

Insert NEW diskette into drive

Press any key to continue.

5. Remove your RCL diskette and place your spare diskette into the disk drive. Close the disk drive door.

6. Press any key to begin the copying process. The system then displays the following message to signal you that copying is taking place:

WORKING!

7. The next message you see is:

Insert ORIGINAL diskette into drive.

Press any key to continue.

Remove your new diskette and place the original into the drive. Close the disk drive door. The system then displays the "WORKING!" message.

8. The system displays these last two messages several times before the copying process is complete. Continue to swap diskettes and press any key until you see the following message, which indicates that this process is complete:

Good copy.

The copying process is now complete.
Insert the disk you want to use now.
Press any key to continue.

9. Insert your copy into the disk drive and press any key to return to the "What would you like me to do now?" prompt.

10. To make additional copies, insert your original into the disk drive and repeat the procedure until you have several copies of your RCL diskette to work with. Keep the original in a safe place, using your copies to write robot tasks.

11. At the prompt, you are ready to proceed to the section of the Tutorial titled "Sample Robot Task" and to begin entering this task.

Once you have finished configuring your diskettes to match your computer system, your screen displays the "What would you like me to do now?" prompt. To make backups of your diskettes:

1. Use the Savvy SUMMON command to display the EXPLANATION of STANDBY TASKS menu again. Enter:

   SUMMON<RET>

   The system then prompts you with the message:

   the standby task

2. Type:

   "<RET>

   and the system displays the following menu:

   EXPLANATION of STANDBY TASKS

   1. ARCHIVE is used to make copies of disks. It can also be used to do special disk related operations.

   2. CONFIGURATOR is used to tell SAVVY what hardware is in your system and to make clean data disks.

   3. DIAGNOSTICS is used to test the integrity of the data on your data disk. It can also clearn the 'your disk may be in trouble' message.

   4. All done with UTILITIES...Go back to 'What would you like me to do now?"

   ENTER the NUMBER of desired action:

3. Choose option 1 by pressing:

   1

   There is no need to press the RETURN key.

   The system then displays the following menu:

-- ARCHIVE MENU --

1) Copy diskettes.

2) Show the diskette drive numbers.

3) Copy between hard-disk and diskettes.

4) Use the SPECIAL disk utilities and diagnostic routines.

5) All done with ARCHIVE...Go back to 'What would you like me to do now?'

ENTER the NUMBER of desired action:

4. Again, choose option 1 by pressing:

1

There is no need to press the RETURN key.

The system then displays the following screen:

COPY DISKETTES

This procedure allows you to copy data from one diskette, called the ORIGINAL, to another one, called the DUPLICATE.

Press ESCAPE (ESC) key to start over.

Do you need more instructions?

5. Answer this question by pressing:

Y

for YES and then follow the instructions as they are presented on the screen.

It usually takes about 90 seconds for the system to copy a diskette. As the copy is being made, the system also verifies the copy. If a copy cannot be verified, a message displays so that you can try again. When a successful copy is made, the screen displays the message:

Good copy

Press any key when ready to continue.

6. After you press another key, you see the message:

```
REMOVE your new copy from Drive #1 then
RESTORE diskettes to original drives.

Press any key when ready to continue.
```

   In fact, you should use your new copies and preserve your original Savvy diskette and your original RCL diskette. So once you have placed your Savvy duplicate in Disk Drive 1 and your RCL duplicate in Disk Drive 2, press any key to continue. The system then displays the ARCHIVE menu.

7. Select option 5 from the ARCHIVE menu to return to the 'What would you like me to do now?' prompt. Press:

```
5
```

   There is no need to press the RETURN key.

You are now ready to move on to the next section, "Sample Robot Task."

C. SAMPLE ROBOT TASK

Now that we have discussed the basics of RCL, let us step through a sample robot task.

For this demonstration, we will assume that you have RCL I and an RB5X with the voice/sound synthesis option.

SCENARIO

> Assume you have invited friends to your home to enjoy snacks and conversation. As an added treat, you want your RB5X to greet each guest on arrival. As each one enters, you press a bumper on the robot, RB5X begins flashing its lights and rolls over to your friend, stops and introduces itself saying, "Hello, I am RB5X: the Intelligent Robot."

Before we actually write this robot task, you should consider the various elements that comprise it: specifically, movement, bumpers sensors, flashing lights, sonar, and voice. The robot task is then broken down into smaller tasks controlling these various elements.

A handy way to organize your approach to this scenario is to use the following format for laying out the logic or sequence of your robot task:

Building Blocks	Programming
Bumper sensor	The robot is programmed to respond to your pressing one of its bumper sensors by flashing its lights, moving, and using its sonar.
Flashing lights	At the touch signal from you, the robot begins to flash its lights.
Movement	At the same time, RB5X begins to move toward your friend.
Sonar sensor Bumper sensor	RB5X is programmed to stop moving as soon as it senses someone with its sonar or touches them with its bumper.
Voice synthesis	The robot greets your guest with the phrase, "Hello, I am RB5X: The Intelligent Robot" as soon as it stops moving.

Now, let's begin to compose our first robot task.

Before your friends arrive:

1. Make sure the Savvy circuit card is in Slot 7 inside your Apple and that the serial card is in Slot 1.

2. Make sure the voice card is in Slot J1 or J2 inside the robot.

3. Insert the RCL diskette, with its label up and the notch to the left, into the Apple disk drive and close the door.

4. Switch the Apple on.

5. When the "What would you like me to do now?" prompt appears, type:

   DEFINE<RET>

6. The next CRT display should be:

   DEFINE a task called

   Let's call this robot task PARTY. So type:

   PARTY<RET>

   NOTE: If at any time what you expect to appear on the screen does not appear, press the ESCAPE key to return to the "Finish, Resume, or Suspend?" prompt. If you then press F (for Finish), you can return to the "What would you like me to do now?" prompt and begin again.

7. The next display should be:

   0--Task PARTY
   1 Does

   where line 0 gives the task title and line 1 is the beginning of what the robot task does. RCL automatically numbers the lines of your robot task.

8. The first thing you must do is prepare the robot's hardware to begin accepting commmands. You need to do this any time you want to write and then download programs from your computer to the RB5X. Type:

   PREPARE THE ROBOT<RET>

9. Next you must let the voice hardware know that it too must be ready to accept commands. Type:

   PREPARE THE VOICE<RET>

10. Next, you must load the command that instructs the robot to introduce itself to your guests. Type:

    X LOAD THE INTRODUCTION <RET>

    The X before this task indicates that it is a sample task that comes on your RCL diskette; you must include the X.

11. Line 4 contains the title of your task and must be included so that when the RB5X reaches the end of the task, the robot will have a point of reference for returning to the start and repeating the greeting again. Type:

    CALL<RET>

    The system answers with:

    this robot task line

Name this line by typing:

"BEGINNING OF PARTY<RET>

Don't forget the first pair of quotation marks. RCL enters the second set for you after you press the RETURN key.

12. Line 5 marks the beginning of the first loop in your robot task. With this general loop, you say to the robot, "You sit and wait patiently until I give you the signal to begin your greeting." Type:

    BEGIN A LOOP<RET>

13. Once you've instructed RB5X to begin a loop, you must also tell the robot how to leave (or exit) the loop. In this case, you want RB5X to sit and wait patiently until you press a bumper. So, type:

    EXIT IF ANY BUMPER TOUCHED<RET>

14. A general loop really consists of three tasks: one that tells it when to start (BEGIN A LOOP), one that tells it what to do within that loop or how to get out of the loop (EXIT IF ANY BUMPER TOUCHED), and one that closes the loop and tells it to repeat the procedure. To program this last task, type:

    REPEAT THIS LOOP<RET>

15. Now, when you touch any bumper on the robot, you want it to begin flashing its lights and to roll up to your guest, using its sonar and bumpers to tell when it has reached your friend. So now type:

    TURN ON THE FLASHING LIGHTS<RET>
    MOVE FORWARD<RET>

16. Here's a good place for another general loop. You want the robot to continue moving forward until either it senses something with its sonar or until it touches something with a bumper. So, type:

    BEGIN A LOOP<RET>

17. This loop has two tasks in it telling it when to stop. Type:

    EXIT IF ANY BUMPER TOUCHED<RET>
    EXIT IF SONAR<RET>

18. After you press the last RETURN, the system prompts you for the sonar distance you wish:

    distance value is less than

This indicates at what distance the sonar is programmed to detect obstacles in the robot's path. In this case, you want the robot to stop just less than a foot from your guests, so type:

```
95<RET>
```

(The value 95 is the minimum distance read by the sonar, and is just less than a foot. For further understanding of sonar values, see the section on "Sonar Machine Code Algorithm" in Chapter D on "Software" in the RB5X Reference Manual.)

19. Once again, close the loop by typing:

```
REPEAT THIS LOOP<RET>
```

20. When either a bumper is touched or the robot's sonar detects an object, you want the robot to stop. Type:

```
STOP ALL MOTION<RET>
```

21. After it stops, you want RB5X to introduce itself. Type:

```
SPEAK<RET>
```

The system then prompts with the message:

```
the phrase called:
```

Enter:

```
"SAY THE INTRODUCTION<RET>
```

This is the phrase you loaded earlier in your robot task with the X LOAD THE INTRODUCTION command. Don't forget the first set of quotation marks. RCL enters the second set for you.

22. Once it has spoken its piece, RB5X should return to its starting point to await another guest. So tell RB5X to turn around by entering:

```
SPIN CLOCKWISE<RET>
```

The system then prompts you with:

```
this many degrees:
```

23. Since you want the robot to turn around and return to the start, type:

```
180<RET>
MOVE FORWARD<RET>
```


24. RB5X should move forward until you press a bumper, so a loop would be useful here. Type:

    BEGIN A LOOP<RET>

25. The robot should continue to move forward until you instruct it to stop, by pressing one of its bumpers. Type:

    EXIT IF ANY BUMPER TOUCHED<RET>

26. Don't forget to close the loop:

    REPEAT THIS LOOP<RET>

27. Once RB5X has returned to the starting point, you need to have it facing front again, so type:

    SPIN CLOCKWISE<RET>

    At the prompt

    this many degrees:

    type:

    180<RET>

28. To have the RB5X turn off its lights, type:

    TURN OFF THE FLASHING LIGHTS<RET>

29. You now want the robot to sit and wait until the next guest arrives. So, to return to the beginning of the entire program, type:

    JUMP<RET>

    At the prompt

    to the line called

    type:

    "BEGINNING OF PARTY<RET>

    Don't forget to type the first set of quotation marks. RCL adds the second.

30. To redisplay the entire task on the screen, type:

    L<RET>

This displays on your screen the RCL robot task listing you see in Appendix 1. (To list only part of the task type the number of the last line you want to see, for example, to list lines 1-15, type 15<RET>.)

31. To signal the end of this robot task, type:

    Q<RET>

    This stands for Quit. The system displays:

    END (Task is 15% full.)

    Once you have ended the task, your screen displays the "What would you like me to do next?" prompt.

32. To load this robot task into the RB5X, switch your robot on (this means the top of the ON/OFF switch is pushed in) and connect the RS-232 cable to the port on the serial communications card in your Apple and Port 1 on the robot.

33. Type:

    BUILD AND LOAD<RET>

The system displays

    Build and Load an RB5X robot task.

34. At the prompt:

    Please enter the name of the robot task:

    type:

    PARTY<RET>

35. The system responds with

    Would you like the task checked for errors? (Y/N)

    Type: Y or N and <RET>

    If you type Y, the system takes a short time to check your program for errors then displays:

    Now building robot task PARTY... One moment, please.

    Note that the process of building and loading the task may take several minutes. You may also use BUILD and LOAD as separate commands; when BUILD is used alone you must enclose the name of the robot task that follows the command (e.g., PARTY) in quotation marks.

The BUILD command begins the process of translating RCL commands into the Tiny BASIC code RB5X understands.

Only one robot task can be translated at a time, and only one robot task at a time remains in the Tiny BASIC code form on your RCL diskette.

For example, you just built the robot task PARTY; it now exists in Tiny BASIC code form on your RCL diskette. If you create another robot task, DANCE, you must build DANCE before you can download it to RB5X, but if you want to reload the robot task PARTY, you do not have to build it again. Therefore, you must build every robot task before you load it into your robot, unless it is the task you last built when you last used RCL.

36. When the system finishes building the task, the system displays the message:

    Ready to load into the RB5X. Press any key to continue.

    After you press a key, the system displays:

    Now loading ... One moment, please

    It also displays a status line indicating the loading of sonar, voice, and the robot task.

    When the computer finishes loading this task into your RB5X, the "What would you like me to do now?" prompt again displays.

37. Disconnect the robot from the RS-232 cable, position it where you want it to wait for your guests, and press any bumper for it to begin its greeting.

38. So now you have a complete robot task in RCL. But suppose you want to change your task? RCL allows you to add, change, or delete lines in your robot tasks using the Savvy EDIT command mode.

    Suppose you want your robot to honk its horn as it begins to approach each of your guests. To have RB5X do this, you need to add a horn-honking command to your robot task, PARTY. Since you just loaded PARTY into your robot, your screen now displays the "What would you like me to do now?" prompt.

    Type:

    EDIT<RET>

    to go into the Savvy edit mode, and at the prompt:

    the

Type:

PARTY<RET>

39. Your screen now displays:

```
0 --- Task PARTY
1 Does
```

which is the beginning of the task listing for PARTY.

Type:

L<RET>

to list the primary tasks that make up this robot task.

40. Looking down the list of primary tasks, you see that line 9, which contains the primary task, MOVE FORWARD, is the line where RB5X begins to move toward your guest. It is after this line that you want to add a command that makes the robot honk its horn.

Type:

9<RET>

Your cursor moves to line 10, and your screen displays the text of lines 0 through 9. You can use this procedure to move to any line in your robot task when you are in the Savvy edit mode.

41. To add the horn-honking command, type:

HONK<RET>

RCL prompts you with:

the horn for this many seconds

42. Type:

3<RET>

You have now added the horn sound to your robot's approach.

43. To view your new robot task PARTY, type:

L<RET>

44. Now type:

    Q<RET>

    to exit the edit mode and return to the "What would you like me to do now?" prompt.

45. Now you can load the new task into your RB5X and run it using the procedures beginning with step 32. Switch RB5X off, then on again or press the software reset button on the interface panel before you BUILD AND LOAD to clear the previous PARTY robot task from its memory.

46. You also use the Savvy edit command mode to delete or change robot task lines as follows:

    When the system displays the "What do you want me to do now?" prompt, type:

    EDIT<RET>

    At the prompt:

    the

    type:

    PARTY<RET>

47. You are now again in the Savvy edit mode. Type:

    L<RET>

    to list your robot task.

48. Let's delete line 10, the HONK command we previously inserted. Type:

    10<RET>

    to move the cursor to line 10.

49. Type:

    D<RET>

    to delete the line.

50. To change a robot task line, you follow the above procedure for deleting the line, then type in the new line and <RET>.

51. To delete an entire robot task, you must delete each of its lines separately. The task name, however, exists on your RCL diskette even after you delete all the task lines. Because you cannot completely remove old tasks from the RCL diskette, we encourage you to reuse old task names or to rename them using the Savvy RENAME command.

52. RENAME is used following the "What do you want me to do now?" prompt. Type:

    RENAME<RET>

    at the prompt:

    old name

    type the name of the robot task you want to change and <RET>.

53. At the prompt:

    to new name

    type the new name of the robot task and <RET>.

To better illustrate the difference between what you wrote in RCL and what RCL told RB5X, we have included in Appendix 1 a copy of the "Party" robot task and the Tiny BASIC translation. As you can see, the RCL text is just 24 lines long -- the Tiny BASIC text stretches to 115 lines of complicated computer code that only machines like RB5X should be forced to learn!

We have also included the detail (which we got by using the Savvy task DETAIL) for each of the primary tasks used to compose the robot task we just wrote. These tasks are listed alphabetically in Appendix 2.

You are now ready to design your own robot tasks for your RB5X. Use the sample scenario form in Appendix 4 to outline the building blocks for your own scenarios and plans.

To start composing own your robot task, follow steps 1 through 8 of the Sample Robot Task using your new scenario data.

D. APPENDICES


```
PARTY (a Task)

 1 Does PREPARE THE ROBOT
 2  and PREPARE THE VOICE
 3  and X LOAD THE INTRODUCTION
 4  and CALL this robot task line "BEGINNING OF PARTY"
 5  and BEGIN A LOOP
 6  and EXIT IF ANY BUMPER TOUCHED
 7  and REPEAT THIS LOOP
 8  and TURN ON THE FLASHING LIGHTS
 9  and MOVE FORWARD
10  and BEGIN A LOOP
11  and EXIT IF ANY BUMPER TOUCHED
12  and EXIT IF SONAR distance value is less than 95
13  and REPEAT THIS LOOP
14  and STOP ALL MOTION
15  and SPEAK the phrase called:  "SAY THE INTRODUCTION"
16  and SPIN CLOCKWISE this many degrees: 180
17  and MOVE FORWARD
18  and BEGIN A LOOP
19  and EXIT IF ANY BUMPER TOUCHED
20  and REPEAT THIS LOOP
21  and SPIN CLOCKWISE this many degrees: 180
22  and TURN OFF THE FLASHING LIGHTS
23  and JUMP to the line called "BEGINNING OF PARTY"
24  and END   (Task is 15% full.)
```

```
Stmt     Tiny BASIC Text                          Meaning of Text

3005      DELAY 1000
3010      NEXT S
3015      RETURN
3100      U=@#7801                                TURN ON A BIT
3105      U=U OR X
3110      @#7801=U
3115      RETURN
3120      U=@#7801                                TURN OFF A BIT @#7801
3125      U=U AND X
3130      @#7801=U
3135      RETURN
3500      V=TOP+500+A:@V=P                        STORE THE PHONEME
3510      A=A+1
3520      RETURN
3600      REM END OF PROGRAM
```

** End of the Program **

```
Stmt    Tiny BASIC Text                          Meaning of Text

 510     P=#0D:GOSUB 3500                        VOICE SUBROUTINE
 520     P=#2A:GOSUB 3500                        VOICE SUBROUTINE
 530     P=#2B:GOSUB 3500                        VOICE SUBROUTINE
 540     P=#35:GOSUB 3500                        VOICE SUBROUTINE
 550     P=#37:GOSUB 3500                        VOICE SUBROUTINE
 560     P=#0E:GOSUB 3500                        VOICE SUBROUTINE
 570     P=#15:GOSUB 3500                        VOICE SUBROUTINE
 580     P=#23:GOSUB 3500                        VOICE SUBROUTINE
 590     P=#2A:GOSUB 3500                        VOICE SUBROUTINE
 600     P=#3F:GOSUB 3500                        VOICE SUBROUTINE
 610     REM LABEL THIS LINE BEGINNING           LABEL THE LINE
 620     REM START A LOOP                        BEGIN A LOOP
 630     Y=@#7800                                TEST FOR BUMPER CONTACT
 640     IF Y<255 GOTO 660                       EXIT IF ANY CONTACT
 650     GOTO 620                                REPEAT THIS LOOP
 660     REM EXIT TO HERE
 670     X=#40                                   TURN ON FLASHING LIGHTS
 680     GOSUB 3100                               GO TURN ON A BIT
 690     @#7802=#09                              GO FORWARD
 700     REM START A LOOP                        BEGIN A LOOP
 710     Y=@#7800                                TEST FOR BUMPER CONTACT
 720     IF Y<255 GOTO 770                       EXIT IF ANY CONTACT
 730     D=0                                     ZERO DISTANCE
 740     LINK #1800                              CALL SONAR
 750     IF D<95 GOTO 770                        EXIT IF LESS THAN VALUE
 760     GOTO 700                                REPEAT THIS LOOP
 770     REM EXIT TO HERE
 780     @#7802=0                                STOP ALL MOTION
 790     S=TOP+500:LINK #2F50:REM SPEAK          SPEAK A PHRASE
 800     @#7802=#05                              SPIN CLOCKWISE
 810     T=2                                     NUMBER OF WHOLE SECONDS
 820     GOSUB 3000                              GO WAIT
 830     DELAY 649                               MILLISECONDS TO WAIT
 840     @#7802=0                                STOP ALL MOTION
 850     @#7802=#09                              GO FORWARD
 860     REM START A LOOP                        BEGIN A LOOP
 870     Y=@#7800                                TEST FOR BUMPER CONTACT
 880     IF Y<255 GOTO 900                       EXIT IF ANY CONTACT
 890     GOTO 860                                REPEAT THIS LOOP
 900     REM EXIT TO HERE
 910     @#7802=#05                              SPIN CLOCKWISE
 920     T=2                                     NUMBER OF WHOLE SECONDS
 930     GOSUB 3000                              GO WAIT
 940     DELAY 649                               MILLISECONDS TO WAIT
 950     @#7802=0                                STOP ALL MOTION
 960     X=#BF                                   TURN OFF FLASHING LIGHTS
 970     GOSUB 3120                               GO TURN OFF A BIT
 980     GOTO 610                                JUMP TO LABEL BEGINNING OF PAR
2999     GOTO 3400                               GOTO END
3000     FOR S=1 TO T                            WAIT T SECONDS SUBROUTINE
```

APPENDIX 1

RCL Robot Task "PARTY" and its Tiny BASIC Translation

Stmt	Tiny BASIC Text	Meaning of Text
10	@#7803=#98	INITIALIZE I/O
20	@#780B=#A7:S=TOP:A=0	PREPARE THE VOICE
30	REM RB5X say the introduction	REMARK
40	P=#1B:GOSUB 3500	VOICE SUBROUTINE
50	P=#02:GOSUB 3500	VOICE SUBROUTINE
60	P=#00:GOSUB 3500	VOICE SUBROUTINE
70	P=#18:GOSUB 3500	VOICE SUBROUTINE
80	P=#35:GOSUB 3500	VOICE SUBROUTINE
90	P=#37:GOSUB 3500	VOICE SUBROUTINE
100	P=#15:GOSUB 3500	VOICE SUBROUTINE
110	P=#00:GOSUB 3500	VOICE SUBROUTINE
120	P=#09:GOSUB 3500	VOICE SUBROUTINE
130	P=#29:GOSUB 3500	VOICE SUBROUTINE
140	P=#3E:GOSUB 3500	VOICE SUBROUTINE
150	P=#3E:GOSUB 3500	VOICE SUBROUTINE
160	P=#2F:GOSUB 3500	VOICE SUBROUTINE
170	P=#00:GOSUB 3500	VOICE SUBROUTINE
180	P=#0C:GOSUB 3500	VOICE SUBROUTINE
190	P=#38:GOSUB 3500	VOICE SUBROUTINE
200	P=#32:GOSUB 3500	VOICE SUBROUTINE
210	P=#15:GOSUB 3500	VOICE SUBROUTINE
220	P=#31:GOSUB 3500	VOICE SUBROUTINE
230	P=#3A:GOSUB 3500	VOICE SUBROUTINE
240	P=#0E:GOSUB 3500	VOICE SUBROUTINE
250	P=#3C:GOSUB 3500	VOICE SUBROUTINE
260	P=#21:GOSUB 3500	VOICE SUBROUTINE
270	P=#29:GOSUB 3500	VOICE SUBROUTINE
280	P=#1D:GOSUB 3500	VOICE SUBROUTINE
290	P=#15:GOSUB 3500	VOICE SUBROUTINE
300	P=#00:GOSUB 3500	VOICE SUBROUTINE
310	P=#29:GOSUB 3500	VOICE SUBROUTINE
320	P=#0F:GOSUB 3500	VOICE SUBROUTINE
330	P=#3E:GOSUB 3500	VOICE SUBROUTINE
340	P=#02:GOSUB 3500	VOICE SUBROUTINE
350	P=#01:GOSUB 3500	VOICE SUBROUTINE
360	P=#19:GOSUB 3500	VOICE SUBROUTINE
370	P=#03:GOSUB 3500	VOICE SUBROUTINE
380	P=#1F:GOSUB 3500	VOICE SUBROUTINE
390	P=#0B:GOSUB 3500	VOICE SUBROUTINE
400	P=#0D:GOSUB 3500	VOICE SUBROUTINE
410	P=#2A:GOSUB 3500	VOICE SUBROUTINE
420	P=#02:GOSUB 3500	VOICE SUBROUTINE
430	P=#00:GOSUB 3500	VOICE SUBROUTINE
440	P=#18:GOSUB 3500	VOICE SUBROUTINE
450	P=#0B:GOSUB 3500	VOICE SUBROUTINE
460	P=#09:GOSUB 3500	VOICE SUBROUTINE
470	P=#1E:GOSUB 3500	VOICE SUBROUTINE
480	P=#1A:GOSUB 3500	VOICE SUBROUTINE
490	P=#02:GOSUB 3500	VOICE SUBROUTINE
500	P=#00:GOSUB 3500	VOICE SUBROUTINE

APPENDIX 2

Primary Tasks Used to Compose Robot Task "PARTY"

PRIMARY TASKS USED TO COMPOSE ROBOT TASK "PARTY"

```
BEGIN A LOOP (a Task)

 1 Does Z COMPILE the BASIC statement "REM START A LOOP" which means "BEGIN A '
OP"
 2  and COPY from "BEGIN" to STRUCTURE TYPE
 3  and SAVE new page in LOOPS
 4  and SAVE new page in STRUCTURES
 5  and END    (Task is 5% full.)


BUILD AND LOAD (a Task)

 1 Does CLEAR the page
 2  and OUTPUT from "*** Build and load an RBSX robot task ***" using format !
AULT
 3  and CARRIAGE return
 4  and CARRIAGE return
 5  and OUTPUT from "Please enter the name of the robot task:  " using format "
FAULT
 6  and INPUT into HERE
 7  and BUILD a program from the robot task HERE
 8  and CLEAR the page
 9  and OUTPUT from "** Build completed, now loading into the RBSX **" using fo
mat DEFAULT
10  and LOAD
11  and END    (Task is 16% full.)


CALL this robot task line <1>  (a Function)

 1 Does PASTE the "REM LABEL THIS LINE " in front of <1>
 2  and Z COMPILE the BASIC statement FRONT which means "LABEL THE LINE"
 3  and PASTE the "L" in front of <1>
 4  and COPY from FRONT to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and END    (Task is 6% full.)


EXIT IF ANY BUMPER TOUCHED (a Task)

 1 Does Z COMPILE the BASIC statement "Y=@*7800" which means "TEST FOR BUMPER
NTACT"
 2  and Z COMPILE the BASIC statement "IF Y<255 GOTO EXIT" which means "EXIT IF
ANY CONTACT"
 3  and COPY from "EXIT" to STRUCTURE TYPE
 4  and SAVE new page in LOOPS
 5  and SAVE new page in STRUCTURES
 6  and END    (Task is 10% full.)
```

```
EXIT IF SONAR distance value is less than <1>  (a Function)

 1 Does Z COMPILE the BASIC statement "D=0" which means "ZERO DISTANCE"
 2  and Z COMPILE the BASIC statement "LINK #1800" which means "CALL SONAR"
 3  and SPLIT the number <1>
 4  and PASTE the "IF D<" in front of INTEGER
 5  and PASTE the FRONT in front of " GOTO EXIT"
 6  and Z COMPILE the BASIC statement FRONT which means "EXIT IF LESS THAN VALU
E"
 7  and COPY from "EXIT" to STRUCTURE TYPE
 8  and SAVE new page in LOOPS
 9  and COPY from YES to SONAR NEEDED
10  and SAVE new page in STRUCTURES
11  and END   (Task is 14% full.)


JUMP to the line called <1>  (a Function)

 1 Does PASTE the "JUMP TO LABEL " in front of <1>
 2  and Z COMPILE the BASIC statement "GOTO GOES HERE" which means FRONT
 3  and PASTE the "J" in front of <1>
 4  and COPY from FRONT to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and END   (Task is 6% full.)



MOVE FORWARD (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#09" which means "GO FORWARD"
 2  and END   (Task is 3% full.)



PREPARE THE ROBOT (a Task)

 1 Does Z COMPILE the BASIC statement "@#7803=#98" which means "INITIALIZE I/O"
 2  and END   (Task is 3% full.)

PREPARE THE VOICE (a Task)

 1 Does Z COMPILE the BASIC statement "@#780B=#A7:S=TOP" which means "PREPARE TH
E VOICE"
 2  and COPY from "YES" to VOICE NEEDED
 3  and END   (Task is 5% full.)



REPEAT THIS LOOP (a Task)

 1 Does COPY from STATEMENT NUMBER to HERE
 2  and Z GET PREVIOUS BEGIN
 3  and DELETE the page indexed by STATEMENT NUMBER from folder LOOPS
 4  and PASTE the "GOTO " in front of STATEMENT NUMBER
 5  and COPY from HERE to STATEMENT NUMBER
 6  and Z COMPILE the BASIC statement FRONT which means "REPEAT THIS LOOP"
 7  and Z COMPILE the BASIC statement "REM EXIT TO HERE" which means ""
 8  and Z PATCH UP EXIT STATEMENTS
 9  and COPY from "REPEAT" to STRUCTURE TYPE
10  and SAVE new page in STRUCTURES
11  and END   (Task is 10% full.)
```

```
SPEAK the phrase called:  <1>  (a Function)

 1 Does BEGIN at page indexed by FIRST in folder SPEECH LOAD DATA
 2  and GET the page indexed by <1> in folder SPEECH LOAD DATA
 3  and COPY from SPEECH ADDRESS to HERE
 4  and PASTE the "S=" in front of HERE
 5  and PASTE the FRONT in front of ":LINK #2F50"
 6  and PASTE the FRONT in front of ":REM SPEAK"
 7  and PASTE the FRONT in front of <1>
 8  and Z COMPILE the BASIC statement FRONT which means "SPEAK A PHRASE"
 9  and END   (Task is 9% full.)


SPIN.CLOCKWISE this many degrees: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by .01472
 3  and    SPLIT the number PRODUCT
 4  and    TAKE this many characters 4 from FRACTION
 5  and    PASTE the INTEGER in front of FRONT
 6  and    GO CLOCKWISE
 7  and    WAIT this many seconds FRONT
 8  and    STOP ALL MOTION
 9  and    END of test
10  and END   (Task is 5% full.)


STOP ALL MOTION (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=0" which means "STOP ALL MOTION
 2  and END   (Task is 3% full.)


TURN OFF THE FLASHING LIGHTS (a Task)

 1 Does Z COMPILE the BASIC statement "X=#BF" which means "TURN OFF FLASHING  I
HTS"
 2  and Z COMPILE the BASIC statement "GOSUB 3120" which means " GO TURN OFF A
IT"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 7% full.)


TURN ON THE FLASHING LIGHTS (a Task)

 1 Does Z COMPILE the BASIC statement "X=#40" which means "TURN ON FLASHING L C
TS"
 2  and Z COMPILE the BASIC statement "GOSUB 3100" which means " GO TURN ON A F
T"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 7% full.)


X LOAD THE INTRODUCTION (a Task)

 1 Does REMARK that: "RB5X say the introduction - new voice cards only!"
 2  and LOAD THE PHRASE called: "SAY THE INTRODUCTION" with phonemes: "H.EH1.EH3
.L.O1.U1.AH1.EH3.I3.Y.PA1.PA1.AE1.EH2.M.THV.UH1.AH1.UH2.ER.B.E1.AY.Y.F.AH1.EH3.Y
.V.PA1.EH1.EH2.K.PA0.S.I1.N.T.EH1.EH3.L.I1.I3.D.J.EH1.EH3.N.T.R.O1.U1.B.AH1.UH .
T.STOP."
 3  and END   (Task is 25% full.)
```

APPENDIX 3

Provided RCL Tasks

```
 TASKS
 Name        (Status)

 ASSIGN A MOTOR CODE
*BEGIN A COUNTED LOOP called <1> beginning at <2> ending at <3>
*BEGIN A LIMITED LOOP
*BEGIN A LOOP
*BUILD a program from the robot task <1>
*BUILD AND LOAD
*CALCULATE variable <1> = <2> (+,-,*,/) <3>  <4>
*CALL this robot task line <1>
 CLEAR ALL ITEMS
*DEFINE A VARIABLE called <1>
 END HERE
 END OF PROGRAM
*EXIT IF ANY BUMPER TOUCHED
*EXIT IF FRONT+REAR BUMPER PRESS
*EXIT IF SONAR distance value is less than <1>
*EXIT IF THE BATTERY IS LOW
*EXIT IF THE CHARGER IS TOUCHED
*EXIT IF THE TAPE IS SENSED
*EXIT IF THIS BUMPER is touched <1>
*EXIT THIS LOOP
*FOLLOW TAPE
*GO CLOCKWISE
*GO COUNTERCLOCKWISE
*HONK the horn for this many seconds <1>
 INITIALIZE MEMORY
 INITIALIZE VARIABLES
*JUMP to the line called <1>
*LIGHTS ROUTINE
*LIST THE PROGRAM
*LOAD
*LOAD THE PHRASE called: <1> with phonemes: <2>
*MAINTAIN CHARGE
*MOVE BACKWARD
*MOVE DISTANCE BACKWARD for this many feet: <1>
*MOVE DISTANCE FORWARD for this many feet: <1>
*MOVE FORWARD
*MOVE FORWARD TIL TAPE NOT SENSED
*MOVE RANDOMLY
*MOVE THE ARM FROM SHOULDER (UP,DOWN,IN,OUT): <1> for this many degrees <2>
*MOVE THE FOREARM (IN,OUT): <1> for this many degrees <2>
*MOVE THE HAND (OPEN,CLOSE): <1> for this (1-100) %: <2>
*MOVE TIMED BACKWARD for this many seconds <1>
*MOVE TIMED FORWARD for this many seconds <1>
*MOVE WITH BETA INTELLIGENCE
 OTHERWISE DO
*PICK A RANDOM DIRECTION
*PIVOT ON LEFT CLOCKWISE
*PIVOT ON LEFT COUNTERCLOCKWISE
*PIVOT ON RIGHT CLOCKWISE
*PIVOT ON RIGHT COUNTERCLOCKWISE
*PREPARE THE ROBOT
*PREPARE THE VOICE
 REMARK that: <1>
*REPEAT THE LIMITED LOOP unless <1> is (<,>,=,<>) <2> to <3>
*REPEAT THIS COUNTED LOOP for the counter <1>
*REPEAT THIS LOOP
 REPORT ASSIGN ERROR
```

```
 TASKS
 Name        (Status)

 REPORT DELETE ERROR
 REPORT GET ERROR  (Undefined)
 REPORT LOAD ERROR
 REPORT MATH ERROR
 REPORT REPLACE ERROR
 REPORT SAVE ERROR
*RUN
*SET the variable <1> equal to <2>
 SHOW ERRORS
 SHOW THE PHONEME DICTIONARY
*SPEAK the phrase called:  <1>
*SPIN AROUND CLOCKWISE this many times: <1>
*SPIN AROUND COUNTERCLOCKWISE this many times: <1>
*SPIN CLOCKWISE this many degrees: <1>
*SPIN COUNTERCLOCKWISE this many degrees: <1>
*SPIN LEFT 90 DEGREES
*SPIN RIGHT 90 DEGREES
 STARTUP
*STOP ALL MOTION
*TEST IF the variable <1> is (=,<,>,<>) <2> compared to <3>
*TURN OFF LED number <1>
*TURN OFF THE FLASHING LIGHTS
*TURN OFF THE HORN
*TURN OFF THE INFRARED LED
*TURN ON LED number <1>
*TURN ON THE FLASHING LIGHTS
*TURN ON THE HORN
*TURN ON THE INFRARED LED
*TURN THE WRIST (CW, CCW): <1> for this many degrees <2>
*WAIT this many seconds <1>
*WAIT RANDOMLY up to this many seconds <1>
*X ALPHA
*X ALPHA W/SONAR
*X BETA
*X BETA W/SONAR
*X BUMPER ACTIVATED MOTIONS
*X CHARGER FINDER
*X LOAD THE INTRODUCTION
*X SIMPLE SIMON
 Z COMPILE the BASIC statement <1> which means <2>
 Z COMPUTE 2 TO THE N-1 , with N = <1>
 Z DELAY for this many loops:  <1>
 Z DO CHECK if item <1> is in folder <2>
 Z DO NUMBER TEST on <1>
 Z EMPTY THE PROGRAM
 Z ERROR CHECK IF/UNTIL STATEMENT
 Z ERROR CHECK MATH FUNCTION
 Z ERROR CHECK STRUCTURES
 Z ERROR CHECK THE FOR STATEMENT
 Z ERROR CHECKING?
 Z FIND the label called <1>
 Z GET PREVIOUS BEGIN
 Z GET PREVIOUS IF OR ELSE
 Z INCLUDE at <1> the statement <2> which means <3>
 Z INCREMENT the <1>
 Z INCREMENT STATEMENT NUMBER
 Z INITIALIZE LEGAL VARIABLES
```

```
TASKS
Name        (Status)

Z INSURE ARM PULSE SUBROUTINE IS
Z INSURE BETA SUBROUTINE IS
Z INSURE END OF PROGRAM IS THERE
Z INSURE PORT 7801 SUBROUTINE IS
Z INSURE RANDOM TURN SUBROUTINE
Z INSURE RANDOM WAIT SUBROUTINE
Z INSURE VOICE SUBROUTINE IS
Z INSURE WAIT SUBROUTINE IS
Z LOAD SONAR
Z LOAD VOICE SUBROUTINE
Z LOOK FOR BEGIN
Z LOOK FOR MATCHING STATEMENT
Z OUTPUT PAGE NUMBER
Z OUTPUT PROGRAM HEADING
Z PATCH UP END STATEMENTS
Z PATCH UP EXIT STATEMENTS
Z PATCH UP JUMP STATEMENTS
Z SAVE FOLDER ERROR
Z SAVE NUMBER OR FOLDER ERROR
Z SAVE STRUCTURE ERROR missing the statement <1> to go with the statment <2> a
statement number <3>
ZZ AVAIL TASK
ZZ AVAIL TASK #1
ZZ AVAIL TASK #10
ZZ AVAIL TASK #11
ZZ AVAIL TASK #12
ZZ AVAIL TASK #13
ZZ AVAIL TASK #14
ZZ AVAIL TASK #2
ZZ AVAIL TASK #3
ZZ AVAIL TASK #4
ZZ AVAIL TASK #5
ZZ AVAIL TASK #6
ZZ AVAIL TASK #7
ZZ AVAIL TASK #8
```

APPENDIX 4

Your Own Application

YOUR OWN APPLICATION

Now that your have used this Tutorial and have seen how to go about writing a robot task for your RB5x, we hope you will use the form below for creating your own robot tasks.

Scenario

Building Blocks	Programming